RSVP

Project cluster: Second Year Project

The IT University of Copenhagen

Group 6:

Martin Jeanty Larsen, mjla@itu.dk, 070678

Nicolai Bo Skovvart, nbsk@itu.dk, 231288

Mark Philip Andersen, mpan@itu.dk, 031088

Ahmad Salim Al-Sibahi, asal@itu.dk, 040191

Anders Bech Mellson, anbh@itu.dk, 310780

May 23, 2011

Preface and introduction

This project has been done on the fourth semester of our bachelors education in software development at the IT University of Copenhagen. We would like to thank Niels Hallenberg for teaching and guiding us through this project. We would also like to thank the group we got to work with from the Singapore Management University (hereafter mainly referred to as SMU) which were Alvin Chua, Lim Cheening, Kang Kai Xin and Gabriel Yee.

Best regards
Martin, Nicolai, Mark, Ahmad and Anders.

This report can be downloaded as a PDF file here `http://mofus.dk/KF02.pdf`
The source code can be downloaded as a ZIP file here `http://mofus.dk/KF02.zip`

Contents

1 Problem explanation and background **4**

2 Requirements specifications **6**
2.1 Server . . . 6
2.2 Client . . . 6
2.3 Web services . . . 7
2.4 Use cases . . . 7

3 data model **10**
3.1 Design decisions on the data model . . . 10

4 Problem analysis **12**
4.1 Server design . . . 12
4.2 Client design . . . 13

5 User manuals and examples **15**
5.1 Login . . . 16
5.2 Create Account . . . 18
5.3 Manage Surveys : Create Survey . . . 20
5.4 Manage Surveys : Take Survey . . . 23
5.5 Manage Surveys : Manage Participants . . . 25
5.6 Account . . . 28

6 Technical Description **30**
6.1 Server . . . 30
6.1.1 Data insertion . . . 31
6.1.2 Data extraction . . . 31
6.2 Client . . . 32
6.2.1 Architecture . . . 32
6.2.2 Authentication and authorization . . . 32
6.2.3 Survey Management . . . 33

7 Test strategy **36**
7.1 Server . . . 36
7.1.1 Unit test . . . 36
7.2 Client . . . 37
7.2.1 Unit test . . . 37
7.2.2 Integration testing . . . 38

8 Documentation of communication **39**
8.1 Internal communication . . . 39
8.2 External communication . . . 40
8.2.1 Agreements from our initial meeting . . . 40
8.3 Reflection on the process of the international collaboration . . . 40
8.3.1 Experiences with used techniques . . . 40
8.3.2 How did we perform as a group . . . 42
8.3.3 Experiences with SMU collaboration . . . 42

9 Recommendations **44**
9.1 System improvements . . . 44
9.2 Code improvements . . . 44
9.2.1 Features . . . 44
9.2.2 Functionality . . . 45
9.2.3 Security . . . 45

10 Conclusion **46**

A Review report **47**

B Explanation of data model **49**

C Meeting Logs and agendas **51**

D F# Assignments **65**
D.1 Ahmad . . . 65
D.2 Mark . . . 68
D.3 Martin . . . 71
D.4 Nicolai . . . 75
D.5 Anders . . . 82

E Responsible persons **87**

F Group constitution **89**

G Web services **93**

H Testing plans **98**
H.1 Server unit tests . . . 99
H.2 Client unit tests . . . 115
H.3 Integration test . . . 123

Chapter 1

Problem explanation and background

Performing surveys is a great way to gather opinions and confirm queries. With the technological improvements of the present such as the internet and wide-spread use of computers, performing surveys have been made easier, faster and significantly cheaper. Online surveys allow users worldwide to answer questions at times that work for them and analyzing the results has become mostly automated saving more time and money.

Software development started of as something that'd usually be done locally, but with the creation of the internet, working globally is also an increasingly seen trend. Global development has been seeing increased exposure, as it potentially allows for development on all 24 hours of the day. Using software development teams in other countries can potentially have a positive economic effect for organizations.

For this project we've been working with both of these areas. We have in cooperation with a software development team in Singapore been developing an online RSVP-system (répondez s'il vous plaît), also known as a survey-system. The Danish team has been developing a web service that allows for a variety of clients to perform surveys, while the Singaporian team has been developing a client to be used with the server.

A client could be created using everything from web-development programming languages such as PHP or ASP to more graphically-oriented solutions made in for example Microsoft Silverlight or Adobe Flash, to regular programs running on desktops or even mobile devices such as phones or tablets. The only requirement for a client is that it can communicate with web services over the internet.

Some goals we've had for the survey-system was for it to be super flexible, allowing a wide range of various clients to present themselves in ways that make sense for that particular client. This has been a great influence in our design of everything from the data model to the exposed web services and where we've placed the responsibility for analyzing data. Because of the need for flexibility, we've decided to create a system where the client is responsible for presenting surveys and the data from completed surveys while the server is responsible for allowing safe manipulation of data.

During the project we've also had other constraints we've worked with. We have, for example, been improving our software development process. We've also been taught the programming language F#, as well as a new programming paradigm, functional programming, which has been used to implement the Server.

Following the cooperation with Singapore, we've also implemented a client of our own. The

client was written in ASP.NET, as the biggest platform for surveys is the online-variety performed in browsers. Surveys such as this can also be performed on most mobile devices, even if this isn't always ideal. The client was made with the idea in mind that taking a survey should be fast and easy, should work in most if not all internet-browsers, and it should be easy to analyze and get an overview of the results in a given survey.

Chapter 2

Requirements specifications

In the following we have listed the requirements for the server and client including the necessary web services and use cases for these.

2.1 Server

- The source code must be written in F# and/or C#
- The server must be running on hotel.itu.dk
- The server must make web services available and allow invocation of the methods described below
- The server must be able to store user and survey data in an MS SQL database corresponding to the data model which ensures functionality to support the web services below
- The server must ensure access rights, such that only authorized users may create, edit and view statistics on surveys

2.2 Client

- The source code must be written in C#
- The client must consist of a user interface which enables the user to call the web services described below
- The client must be able to present data to the user according to the use cases described below
- The client must have a high level of usability
- It must be possible to compile and run the client program from a laptop with VS 2010 including F# support installed and a browser

2.3 Web services

- The web services must be written in the ASP.NET framework
- The web services must have methods supporting:
  - Creation/deletion of survey hosts
  - Authentication of a host
  - Changing a host password
  - Creation/deletion of surveys
  - Adding/removing participants to/from a survey
  - Retrieving participants for a given survey
  - Retrieving surveys for a given host
  - Creation/deletion of guests lists
  - Retrieving defined guests list for a host
  - Retrieving participants in a given guest list
  - Adding/removing participants to a guest list
  - Viewing participants for a given survey
  - Retrieving the questions from a given survey
  - Responding to a survey question
  - Checking that all mandatory questions for a survey has been answered
  - Editing the response to a survey question
  - Retrieving results for a given survey

2.4 Use cases

- Create/Delete Host
  - Client calls CreateHost/DeleteHost WS with email and password
  - WS invokes CreateHost/DeleteHost function in Host module
  - Controller performs DB query needed to create or delete the host
  - WS returns true if everything went well, otherwise an exception is thrown with a message about what went wrong
- Authenticate Host
  - Client calls AuthenticateHost WS with email and password
  - WS invokes authenticateHost function in the Utility module which queries the DB to check if the passed parameters corresponds to a valid host and the password is correct
  - WS returns true if the host authenticates, otherwise an exception is thrown with a message about what went wrong
- Create Survey

  - Client has filled out the create survey form and calls CreateSurvey WS with email, password and survey contents in the Template language
  - WS invokes CreateSurvey function in Survey module including submitted data
  - The function parses the data into SQL which is written to the Database
  - WS returns the Id of the created survey to the Client if it went well otherwise an exception is thrown with a message about what went wrong

- Delete Survey
  - Client calls DeleteSurvey WS with surveyId, username and password.
  - WS invokes deleteSurvey function in the Survey module which queries the DB to check if the survey exists and the supplied user authenticates and is host for the survey
  - WS returns true if everything went well, otherwise an exception is thrown with a message about what went wrong

- Add/Remove Invitee to Survey
  - Client calls AddInviteeToSurvey/RemoveInviteeFromSurvey WS with survey Id, invitee email and host email/password
  - WS invokes addInviteeToSurvey/removeInviteeFromSurvey function in the Survey module which performs the query to the DB which adds/removes the invitee from the survey
  - WS returns true if everything went well, otherwise an exception is thrown with a message about what went wrong

- View Invitees From Survey
  - Client calls ViewInvitees FromSurvey WS with survey Id and host email/password
  - WS invokes viewInviteeFromSurvey function in the Survey_Invitee module which fetches the participants from the DB if the host authenticates
  - WS returns and array of Invitees if everything went well, otherwise and exception is thrown with a message about what went wrong

- Get Survey Questions
  - Client calls GetSurveyQuestions WS with survey Id
  - WS invokes getSurveyQuestions function in Survey module which fetches the data for the given survey from the database
  - WS returns a string containing the survey questions if everything went well, otherwise an exception is thrown with a message about what went wrong

- Get Survey Results
  - Client calls GetSurveyResults WS with survey Id and host email/password
  - WS invokes getSurveyResulst function in Survey module which fetches the data for the given survey from the database if the host authenticates
  - WS returns a string containing the survey results if everything went well, otherwise an exception is thrown with a message about what went wrong

- View Surveys
  - Client calls ViewSurveys WS with host email/password
  - WS invokes viewSurveys function in Survey module which fetches the ids of surveys for the supplied host if he authenticates
  - WS returns an array of Surveys if everything went well otherwise an exception is thrown with a message about what went wrong
- Submit/Remove Question Answer
  - Client calls SubmitQuestionAnswer/RemoveQuestionAnswer WS with optionId for the question, invitee email and answer (for the first of the two)
  - WS invokes submitQustionAnswer/removeQuestionAnswer function in the Survey_Answer module which ensures that the invitee is valid and is part of the survey which contains the question
  - WS returns true if everything went well otherwise and exception is thrown explaining what went wrong
- Create/Delete Guest List
  - Client calls CreateGuestList/DeleteGuestList WS with host email/password and the name of the guest list
  - WS invokes createGuestList/deleteGuestList function in GuestList module which performs the query to the DB if the host authenticates
  - WS returns true if everything went well otherwise an exception is thrown explaining what went wrong
- Add/Remove Invitee To/From Guest List
  - Client calls AddInviteeToGuestList/RemoveInviteeFromGuestList WS with guest list name, host email/password and the invitees email (+name for AddInvitee)
  - WS invokes addInviteeToGuestList/removeInviteeFromGuestList function in the GuestList module which performs the query to the DB if the host authenticates
  - Returns true if everything went well, otherwise an exception is thrown with a message explaining what went wrong
- View Guest Lists
  - Client calls ViewGuestLists WS with host email and password
  - WS invokes viewGuestLists function in the GuestList module which fetches the names of the guests lists for the supplied host if he authenticates
  - Returns an array with strings containing the guest list names if everything went well, otherwise and exception is thrown with a message explaining what went wrong
- Change Host Password
  - Client calls ChangeHostPassword WS with host email/password and the new password
  - WS invokes changeHostPassword function in the Host module which performs the query to the DB if the host authenticates
  - Returns true if everything went well, otherwise an exception is thrown with a message explaining what went wrong

Chapter 3

data model

When we had to choose a domain for our RSVP project, we chose the survey domain. In order for us to get a great system we started by looking at other survey systems. The online service SurveyMonkey seemed to have all the functionality we were after, it has been a great inspiration on what a good survey system must include. When designing our data model we used this inspiration to make sure we could support all the functionality in our model.

3.1 Design decisions on the data model

Our process towards agreeing on a data model has been very democratic. We used a whiteboard for drawing sketches for each other. Then we stared at the solution for a while and if we didn't see any errors we moved on. If something was wrong, we drew another sketch and continued doing this. After some iterations and discussions we finally came up with our final model as seen below.

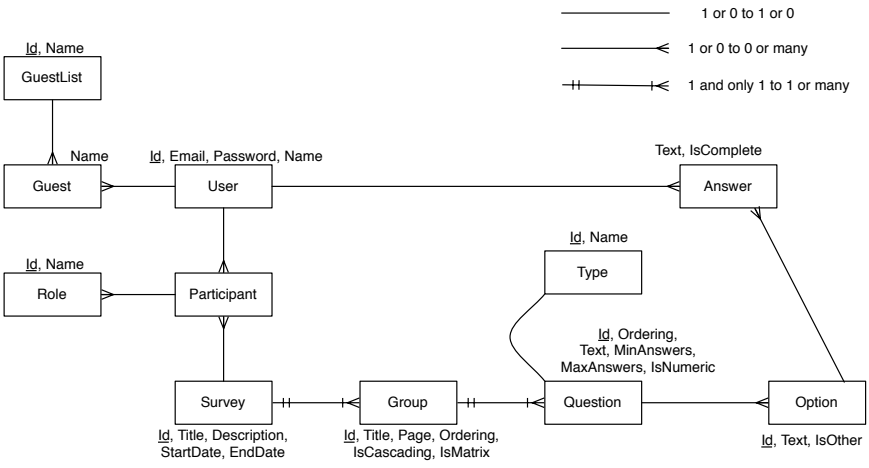


Figure 3.1: Our final data model, explanation in appendix B on page 49.

Our main problem designing the data model was how we could support matrix questions. The problem with matrix questions is that it is actually several questions. So how could we store

this in the data model as a single question? We ended up supporting it with groups. Our group entity has a boolean called IsMatrix. If this is set to TRUE then the client knows that it should represent this group as a matrix type question[1].

We had to implement two extra entities in our model after talking to our co-students at SMU. They wanted each user to be able to have different names in different surveys. Also they wanted names on the list of users receiving a given survey.

The way we solved this was by adding GuestList and Guest to our data model. It didn't break any functionality, it only provided the extra functions they needed for their client.

Another thing we discussed and reworked a few times before getting it right, was how to handle different user permissions. We wanted to have different types of users in the system. A user which could create, modify and send out surveys - an admin. We also needed normal users who would receive a survey and give their answers. We thought about having an enum type which would hold the role for each user. That didn't work out, because we wanted to be able to have different permissions / roles on each survey. What we did was to have a connection, participant entity, from user to survey that had another connection to a role entity. That solved our problem, because a role is dependant on participation in a survey.

[1] We didn't manage to implement this in client.

Chapter 4

Problem analysis

4.1 Server design

The purpose of the server would be to handle all the program logic required for a survey system to function. The preliminary requirements was how the communication between the server and client is handled, this had to be done using Web service which makes use of SOAP (Simple Object Access Protocol). The program logic for the server had to be written in the .NET framework using the language F#. F# does not natively support exposure of web services but as were using the .NET framework it was easy to expose them using C#.

Looking at the .NET framework, this presented us with two major frameworks for supporting SOAP web services over HTTP. The two major frameworks that support exposure of web services are ASP.NET and WCF (Windows Communication Foundation).

Comparing the two, ASP.NET web services are designed to be used in conjunction with ASP.NET and is essentially a simplistic approach to working with web services under ASP.NET. WCF is a general-purpose framework that aims to provide information exchange between applications in the .NET framework.

Our choice fell on using ASP.NET web services as it felt like the simplest and cleanest solution that suited our needs for providing the information exchange between the server and the client.

A key design decision arose when discussing on how we wanted to represent the survey information that goes back and forth between the client and server, this included in general information about a complete survey and answers to questions in the survey. For this we needed a way to group this information and send it back in structured manner. A data format called XML came to mind. XML (eXtensible Markup Language) is used in tons of applications and is for example used by the SOAP protocol to represent data. Another way of sending the information back and forth between the client and server could be to send a serialized object back which contained the information in some state. Further options includes HTML/JavaScript. HTML was a potential candidate, but it could lead to problems with differentiating the responsibilities of the client and server. We wanted a clear separation of client and server, and we didnt want the server to worry about how the client was going to represent the data it is sent. We hoped to achieve a more data-oriented structure by doing this. Other negative aspects would include harder maintainability and updatability, due to tighter coupling with the client.

Based on these thoughts, we decided to use XML to represent survey data as it suited our needs by not telling how data should be represented in the client.

A second key design decision arose when talking about security. For many of the web service calls that go back and forth between the the server and client, an identifier is needed to determine

what rights and so on a user has to the specific call. This of course includes a password. A password in its pure form is simply a combination of letters, signs and numbers. To avoid having the password just appear like a string of text in the database, we wanted a way to obfuscate this information. The way this is done is often by performing hashing on the password. The way hashing works is by transforming an input of data into a fixed sized bit string. This is a one-way deal, the only way you could get the original data is by hashing that same data and comparing the hash values. The way it will work is that the hash value generated from the password will be stored in the database, and authentication will work by comparing the hash value of the password in the database with the hash value of the one provided. This opens up for a security risk, as it will be prone to hash dictionary attacks. However there is exists measures to counter that type of attack. Briefly, this is done by attaching what is called a salt value to the input and hashing it all together, thus requiring both the password and salt value to be guessed in order to crack the password. There are many alternatives for encrypting data, however in this case, hashing the password together with a salt value, provides strong security and is what we have chosen to encrypt our passwords.

Moving on to the communication between the program and the database, we ran into a couple of choices. The preferred way to handle communication from within the F#(or .NET in general) domain logic was to use LINQ (Language Integrated Query) which we were introduced to. Previously, working in java, we had used normal SQL by performing SQL-string based queries to query the database. Performing queries to this is prone to a type of security attack referred to as SQL-Injections, which can ruin a database completely. LINQ is introduced as an integrated component on the .NET platform, and exists to make working with databases from within the .NET platform a much easier and safer experience. As opposed to performing SQL-string based queries, one of the features that LINQ offers is type safety, and also security from SQL-injection attacks. LINQ offers two main variations which were interesting to us. LINQ-to-SQL and LINQ-to-Entities. Firstly, LINQ-To-SQL is whats called an ORM (Object-relational mapping) framework, which provides direct 1-1 mapping from an MS SQL database to .NET classes, in terms of compatibility it is a MS SQL server exclusive feature. LINQ-to-Entities is similar but has a few differences. LINQ-to-Entities is part of the ADO.NET Entitiy framework and as opposed to LINQ-to-SQL, LINQ-to-Entities is not exclusive to MS SQL server only. This allows for a wider compatibility range, if you e.g. should choose to change to a different database. It also offers easy generation of database entities from within Visual Studio, as it is very integrated. For out solution we chose LINQ-to-Entities, at it provided an easy platform for us to develop our database on.

4.2 Client design

For the client there were no real restrictions on how we should present the data provided by the server. We had free hands for choosing whether we wanted the client to e.g. be a normal application or an application hosted on the web.

The main purpose of the client is simply to present information provided by the server, and send information back to it again, for example the survey program logic is entirely handled by the server.

Looking at how surveys are offered electronically today, they are nearly all hosted on the web and this makes perfect sense as this allows for the most people to be able to participate in the surveys without the need for e.g. running a dedicated application on each computer. So the two main choices for the client was to either present it as an application, which would run on each survey participants computer or to present it on the web providing wide availability for the

users.

Looking at the choice of having a dedicated application on each computer what jumps to mind as the easiest approach to this would be using .NET Windows Forms. Windows forms is widely used when developing applications under the .NET framework, it's based on C# and is very easy to work with.

Having a look at the solution for having the client hosted on the web presents a wider variety of options. A new promising feature under .NET is Microsofts Silverlight. Silverlight is designed to try and compete with Adobe Flash. Silverlight is based on the WPF (Windows Presentation Foundation) framework, which can been seen as a successor or an upgrade to the older Windows Forms. Siverlight "feels" like a normal application you would have running on your computer, but the key difference is that it is hosted on the web instead, the framework is easy to use if you already have experience with developing applications using Windows Forms. Some other options include ASP.NET, which is essentially web pages that utilizes the .NET platform, and thus can make use of all the .NET features and languages like C#.

Our choice on what we wanted the client to be fell on web-based solution, under ASP.NET, as this appealed to the majority of the group. ASP.NET also provides a more managed design-oriented approach for development webpages called ASP.NET MVC, and having already coded our web services in ASP.NET this seemed like a natural choice. MVC incorporates the Model/View/Controller design pattern, whose purpose is to separate the programs domain logic from the user interface, providing easier extendibility and overview. Testing is also made easier when developing using this pattern as the components can be tested seperately.

Chapter 5

User manuals and examples

In this section we have created user manuals to help users understand how the system works and at the same time provide walkthroughs of the various parts of the system.

The RSVP system can be reached at `http://hotel.itu.dk/Rsvp06_1/`.

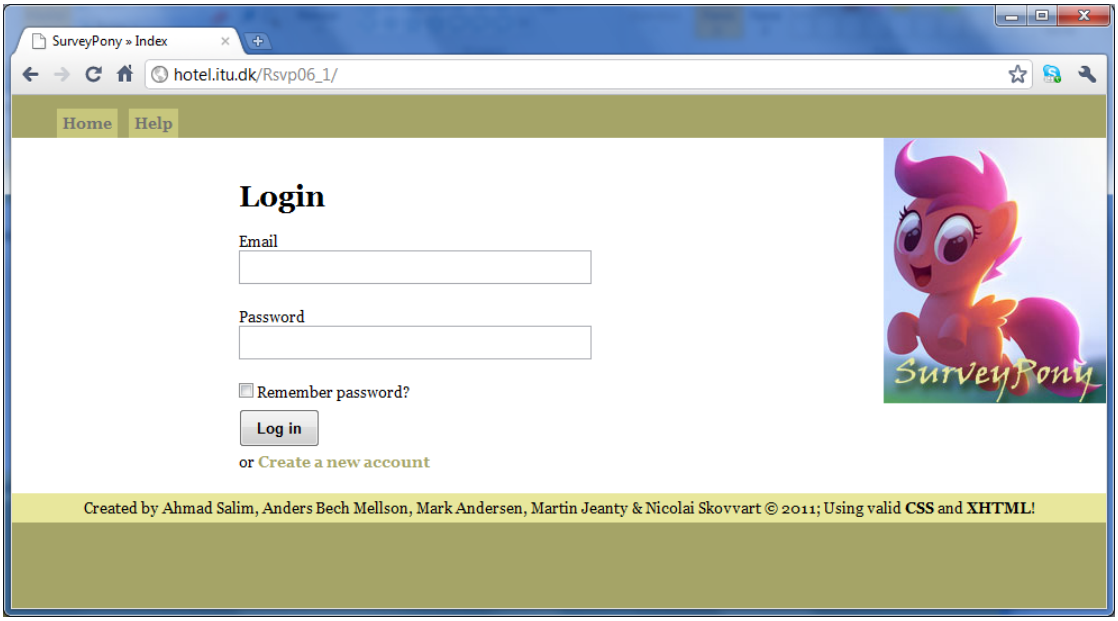
The user manuals covers the following sections and functionality:

- Login: How to login to the RSVP system
- Create Account: How to create an account so that you can login and start using the RSVP system
- Manage Surveys, Create Survey: Walkthrough of creating surveys
- Manage Surveys, Take Survey: Walkthrough of answering surveys
- Manage Surveys, Manage Participants: Walkthrough of managing participants and sending out invitations for surveys
- Account: How to change account password or delete an account

5.1 Login

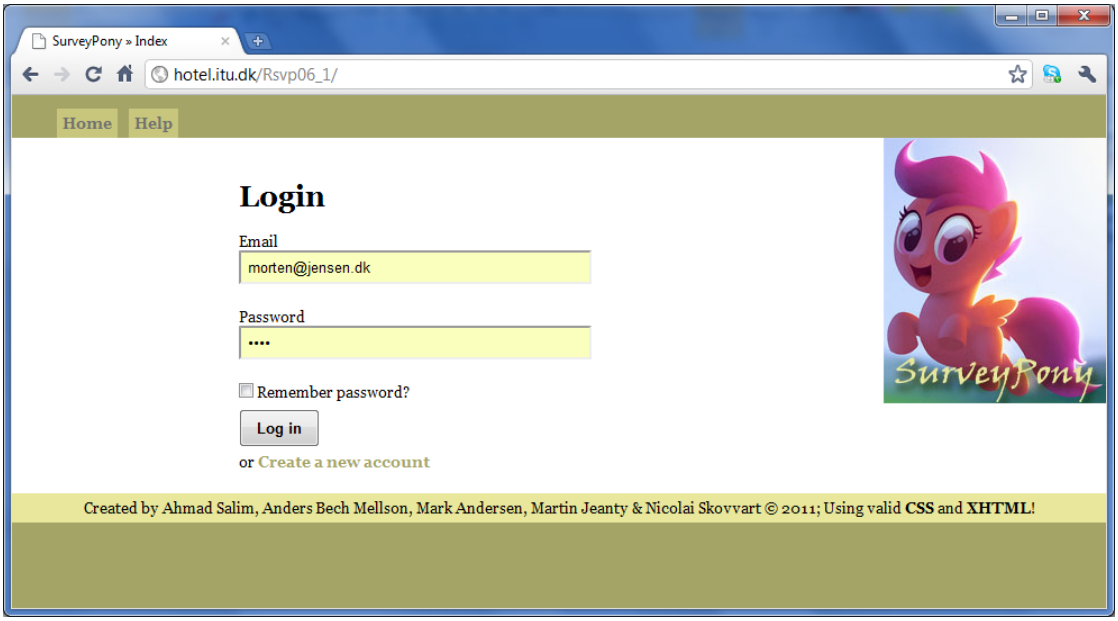
The following manual describes how to login to the RSVP system.

1. The first screen you meet when entering the survey system is the login screen.

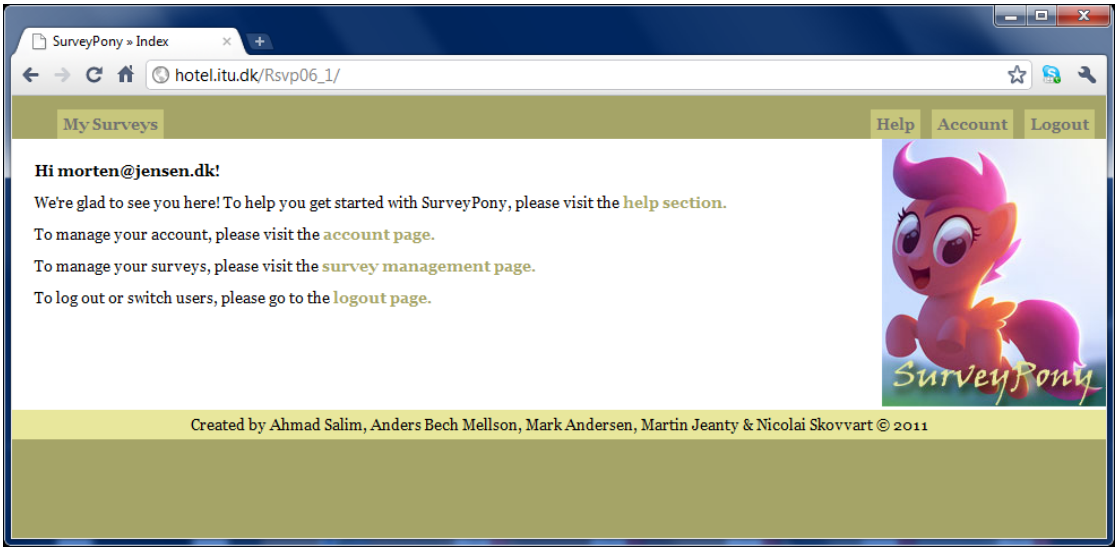


2. Here you must type your email address and password and you have the option of saving the password so you don't have to login every time you visit the RSVP system (default timeout is 30 minutes).

3. Click "Log in". **Note:** If you don't have an account already, see the "Create Account" section for help.



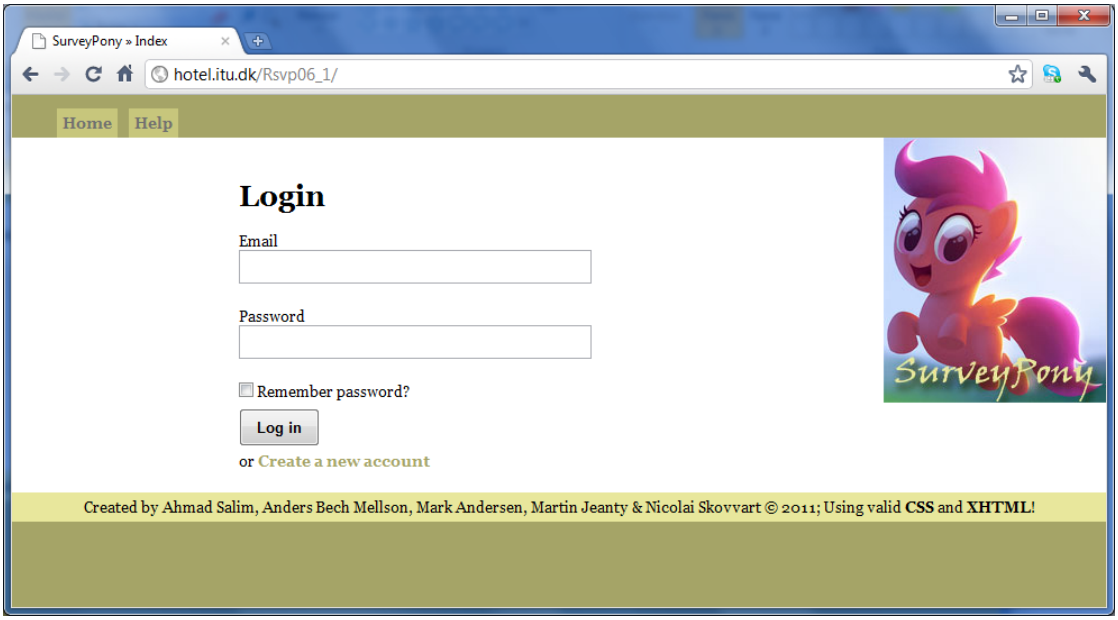
4. Given you've entered a valid email address and corresponding password you're now logged in and should a page similar to the following example


SurveyPony » Index
hotel.itu.dk/Rsvp06_1/
My Surveys
Help
Account
Logout
Hi morten@jensen.dk!
We're glad to see you here! To help you get started with SurveyPony, please visit the help section.
To manage your account, please visit the account page.
To manage your surveys, please visit the survey management page.
To log out or switch users, please go to the logout page.
SurveyPony
Created by Ahmad Salim, Anders Bech Mellson, Mark Andersen, Martin Jeanty & Nicolai Skovvart © 2011

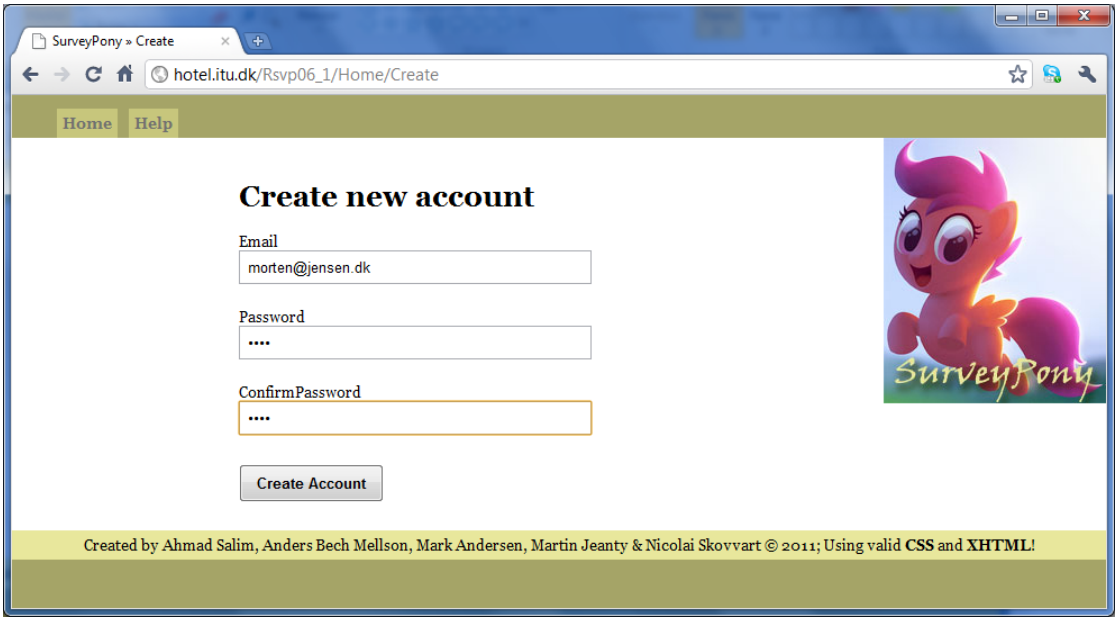
5.2 Create Account

The following section describes how to create a new account in the RSVP system so that you can start creating surveys.

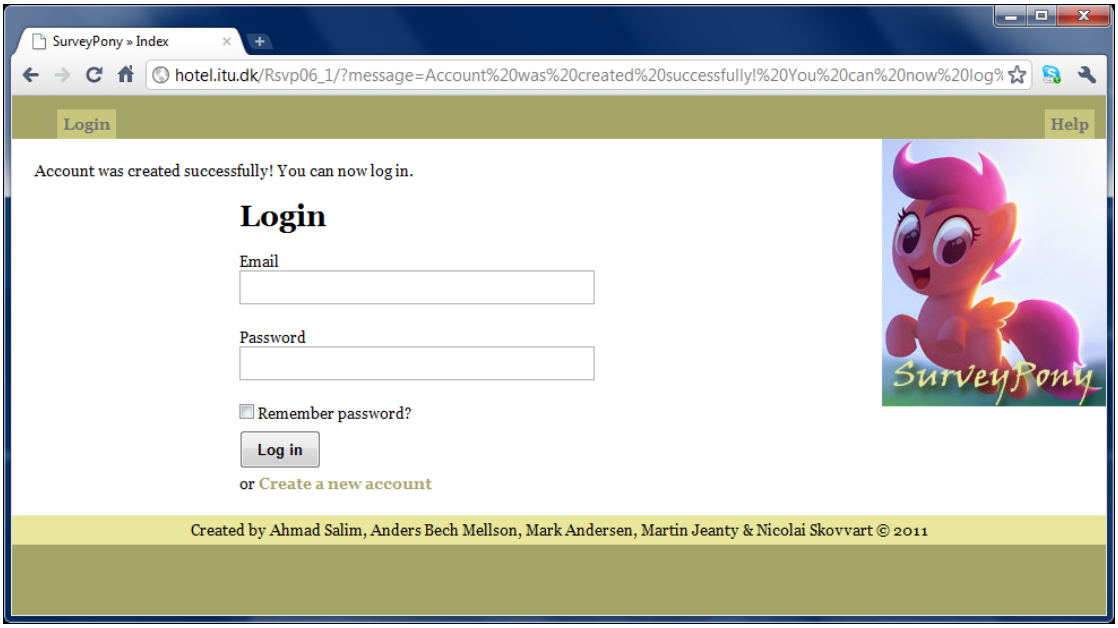
1. If you don't have an account already you will have to click the "Create a new account" link on the frontpage.



2. Type in a valid email address and the password you wish twice to confirm, finish with "Create Account".



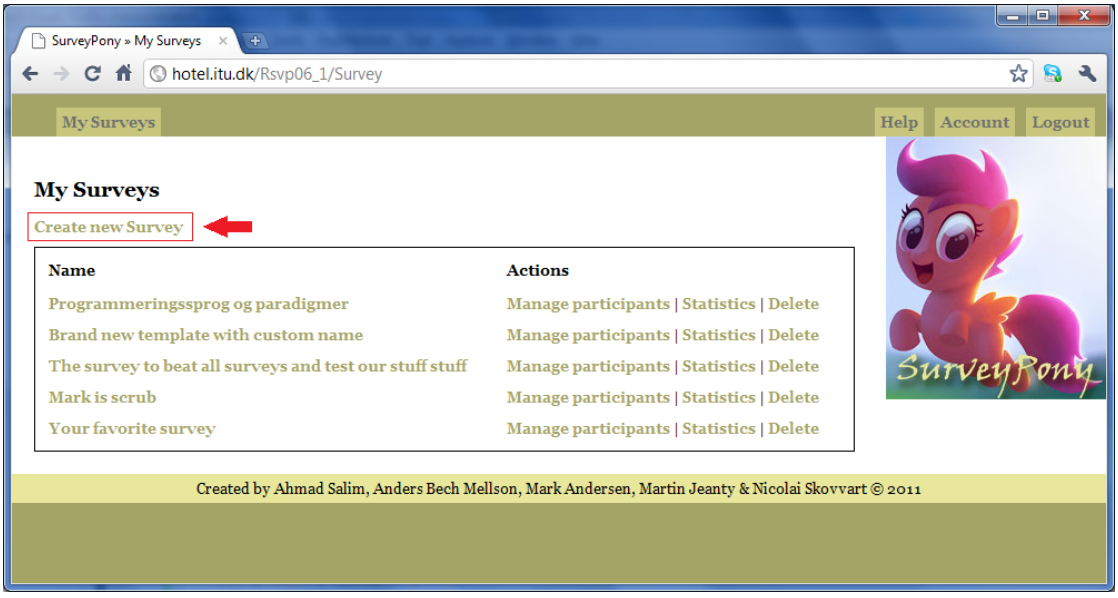
3. Given you've typed a valid email address your account has now been created and you should see the following page.



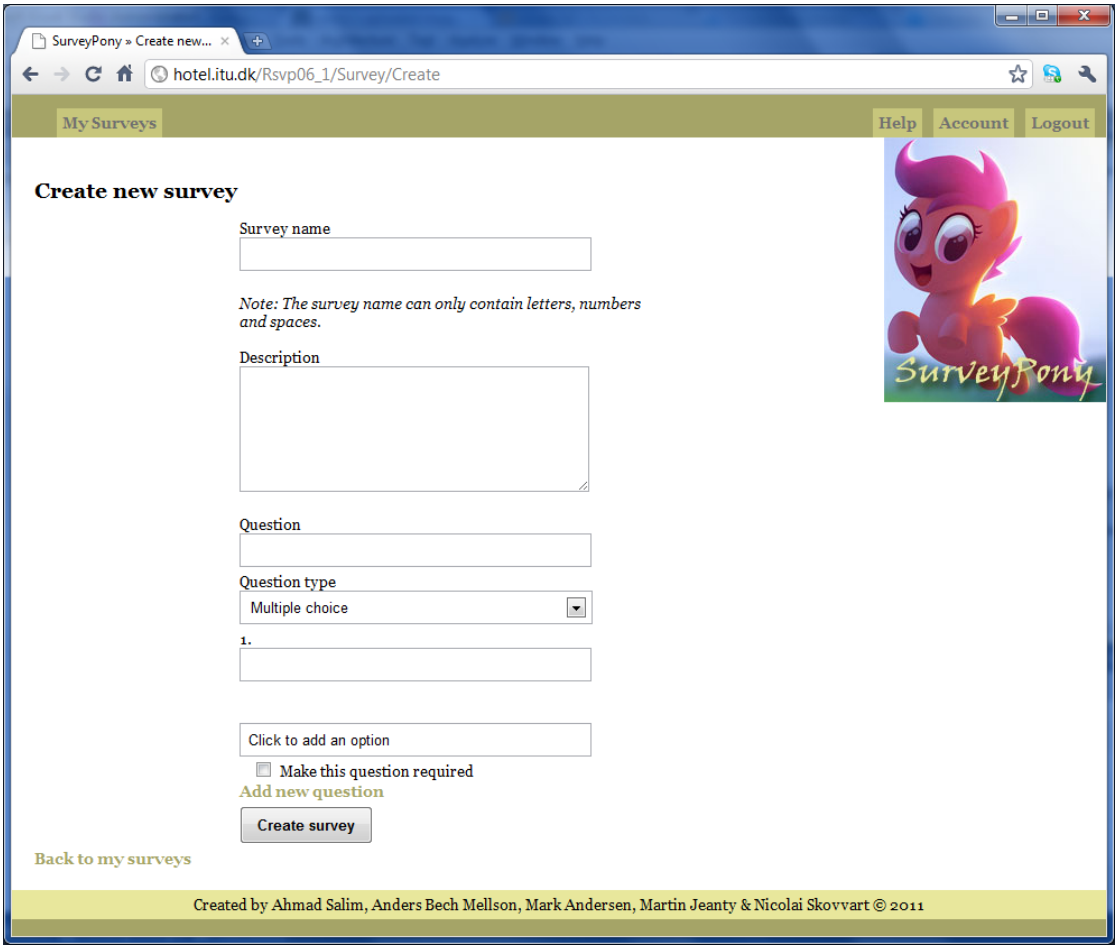
5.3 Manage Surveys : Create Survey

This manual explains how to create a survey and specifies the different types of questions in the RSVP system.

1. From the "My Surveys" menu you can create a new survey by clicking the "Create new survey" link.


SurveyPony » My Surveys
hotel.itu.dk/Rsvp06_1/Survey
My Surveys
Help
Account
Logout
My Surveys
Create new Survey
Name
Actions
Programmeringssprog og paradigmer
Manage participants | Statistics | Delete
Brand new template with custom name
Manage participants | Statistics | Delete
The survey to beat all surveys and test our stuff stuff
Manage participants | Statistics | Delete
Mark is scrub
Manage participants | Statistics | Delete
Your favorite survey
Manage participants | Statistics | Delete
SurveyPony
Created by Ahmad Salim, Anders Bech Mellson, Mark Andersen, Martin Jeanty & Nicolai Skovvart © 2011

2. You're now in the survey builder screen where you setup your survey.



3. Setting up your survey has the following steps:

- Add a name for your survey
- Add a description to your survey
- Add one or more questions to your survey

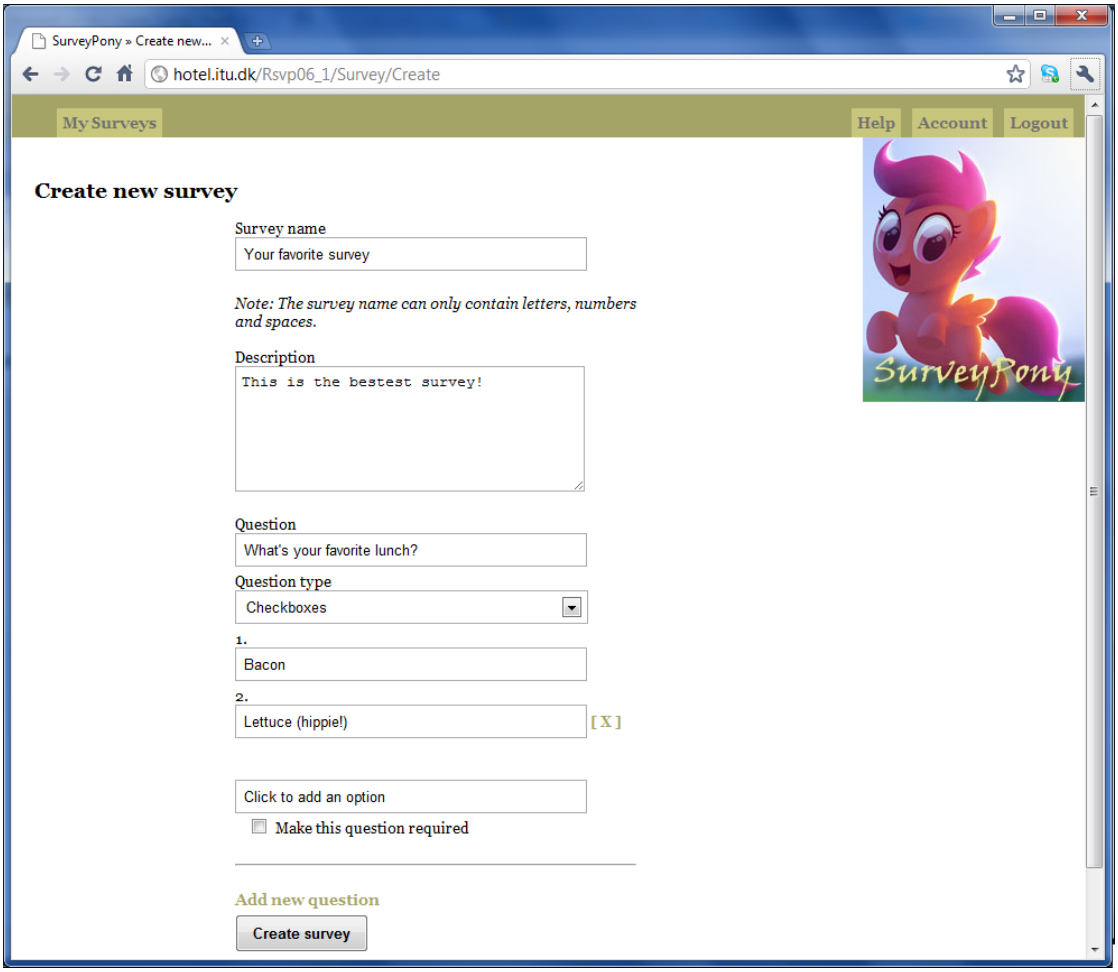
For each question you add to your survey (minimum one) you will have to:

- Define the question to be answered
- Define the type of question among the following available types
  - Checkbox: multiple answers are allowed by ticking as many or as few checkboxes, corresponding to each of your predefined questions, as the user wants
  - Multiple choice: Also known as radio buttons - for this type of question only one option may be selected as the answer of your predefined answer options
  - Text: This type of question is answered by typing in text in a field. **Note:** This type of question is not very useful for statistics

  - Explanation: If you need to add further explanation to a question you can use this type

- For each question you have the option of selecting if it's a required question. This will force users that are taking your survey to answer the question(s) marked as required
- For the Multiple choice and Checkbox type questions you can click the "Click to add an option" button for every new answer option you wish to add to this question

When you're done with your question you can add more questions by clicking the "Add new question" link.

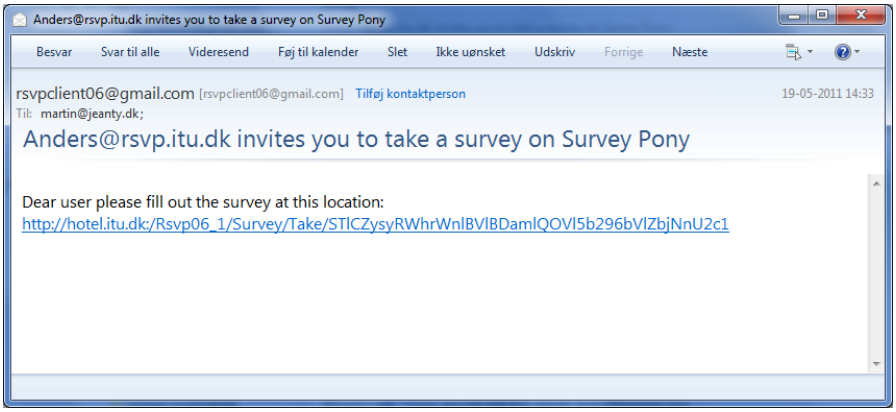


4. Finish creating your survey by clicking the "Create survey" button.

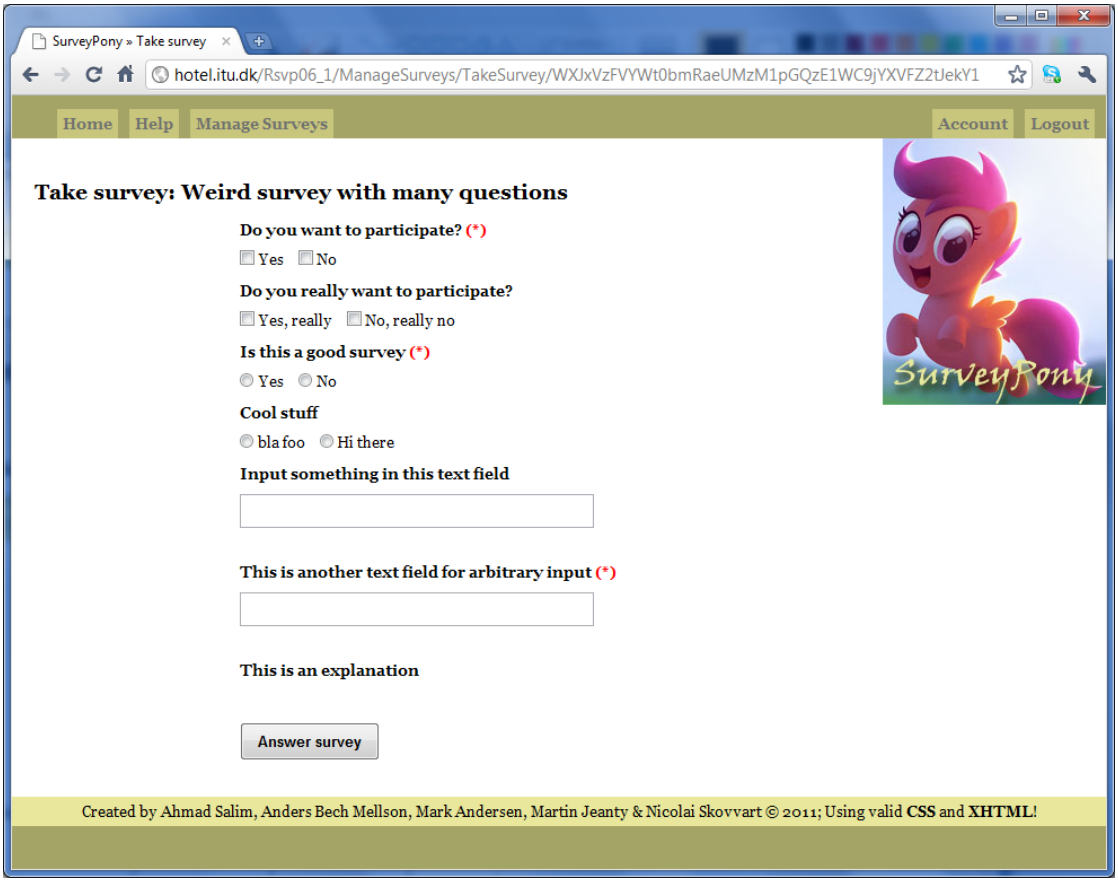
5.4 Manage Surveys : Take Survey

This manual describes how you respond to a survey invitation.

1. If you have been invited to participate in a survey, you should have recieved an email resembling the following example. Click the link in the email to open the survey answering page.

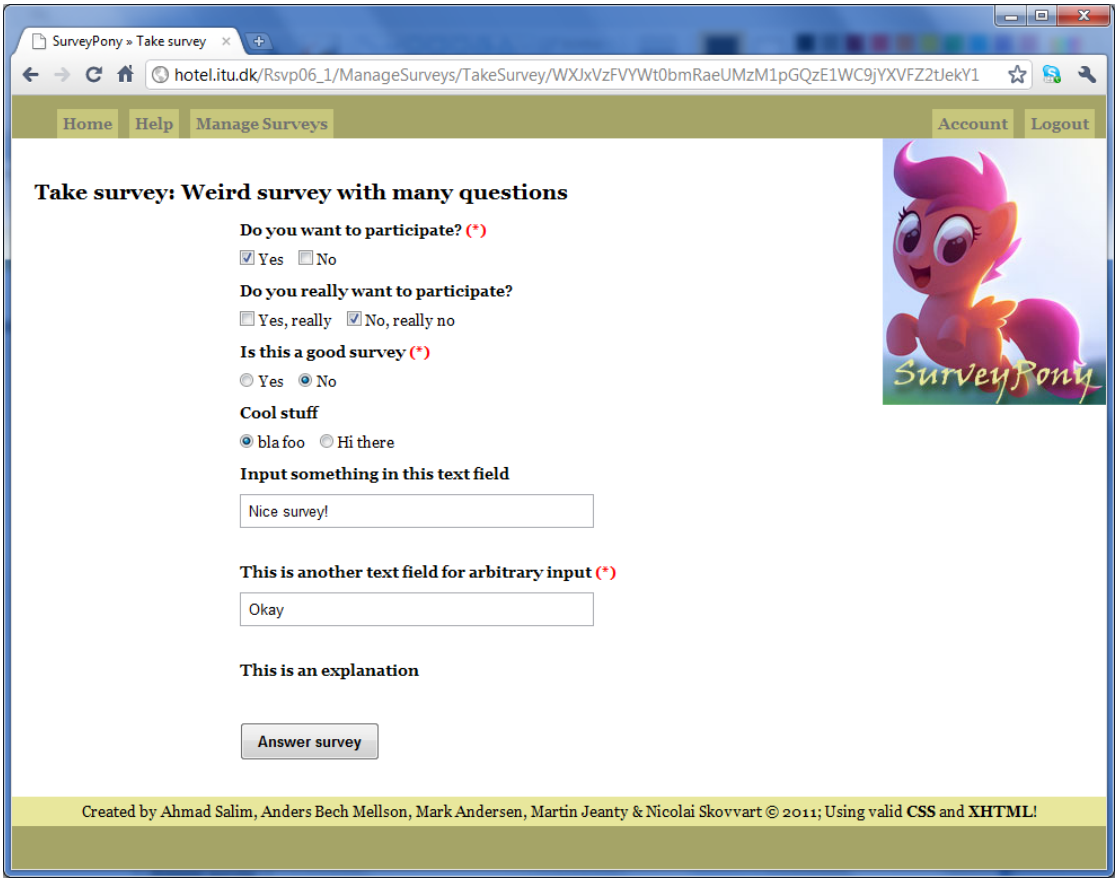


2. Example of a survey.



3. Fill out the survey as in the example below, finish by clicking the "Answer survey" button.

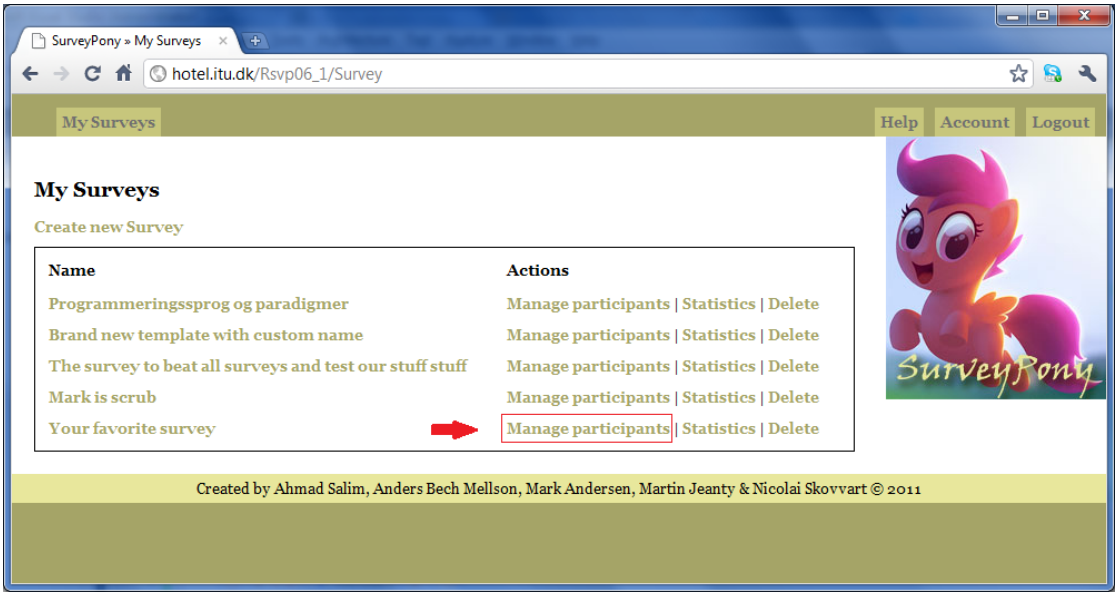
Note: Questions marked with a red asterisk (*) are questions that you are required to answer. If you don't answer one or more of these the survey submission will fail.



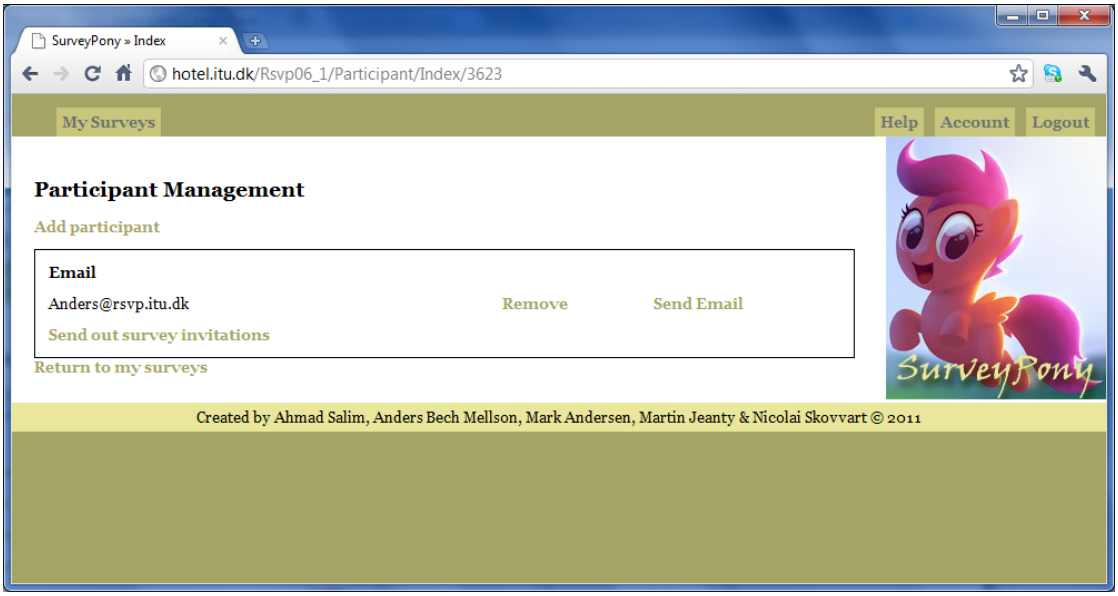
5.5 Manage Surveys : Manage Participants

This manual explain how to add/remove participants to your surveys and sending out survey invitations.

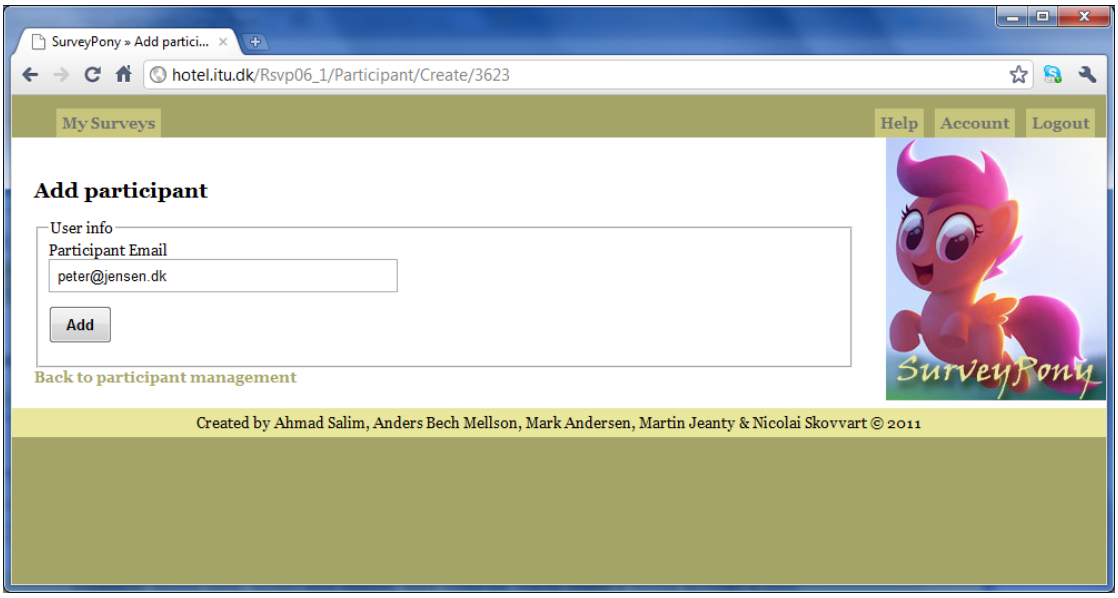
1. From the manage surveys screen you can select "Manage participants" to add or remove participants and send out survey invitations for your surveys.


SurveyPony » My Surveys
hotel.itu.dk/Rsvp06_1/Survey
My Surveys
Help
Account
Logout
My Surveys
Create new Survey
Name
Actions
Programmeringssprog og paradigmer
Manage participants | Statistics | Delete
Brand new template with custom name
Manage participants | Statistics | Delete
The survey to beat all surveys and test our stuff stuff
Manage participants | Statistics | Delete
Mark is scrub
Manage participants | Statistics | Delete
Your favorite survey
Manage participants | Statistics | Delete
SurveyPony
Created by Ahmad Salim, Anders Bech Mellson, Mark Andersen, Martin Jeanty & Nicolai Skovvart © 2011

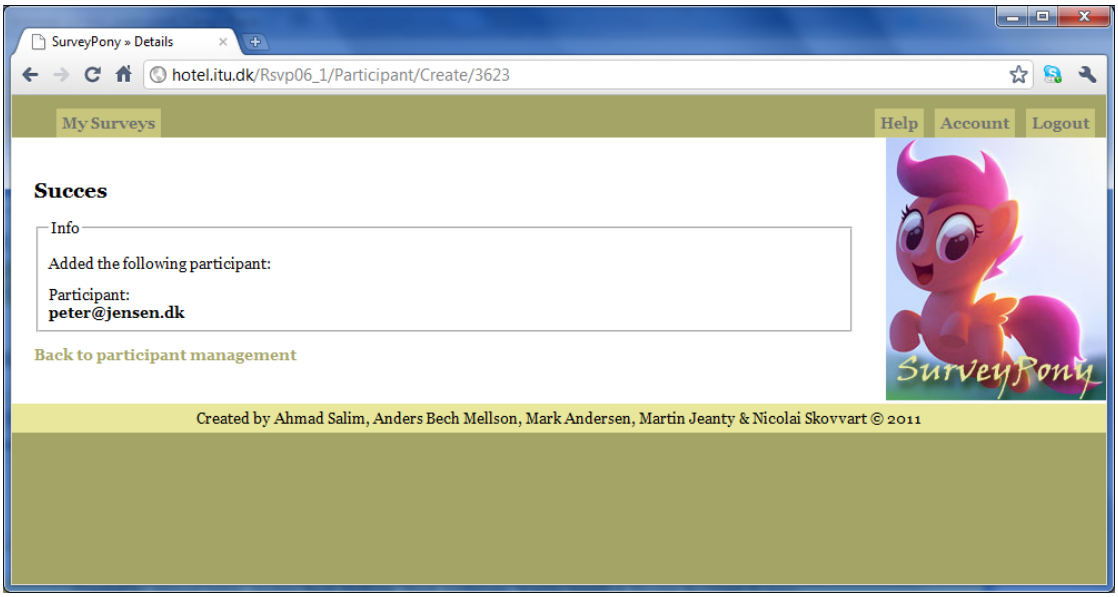
2. Click the "Add participant" link to add a new participant to your survey.



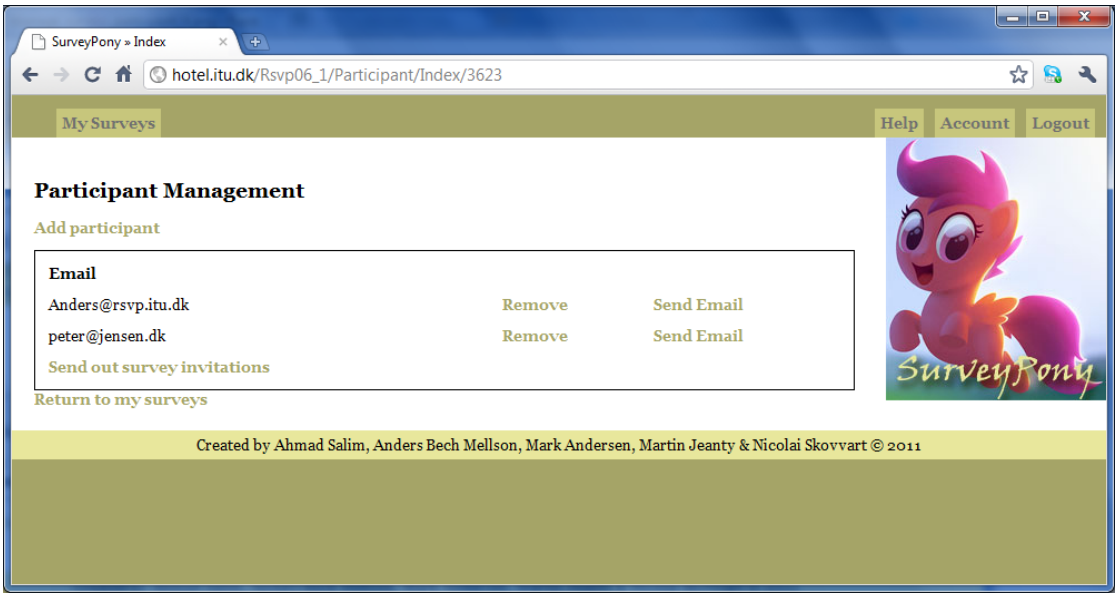
3. Enter the email address of the participant you wish to add and click "Add".



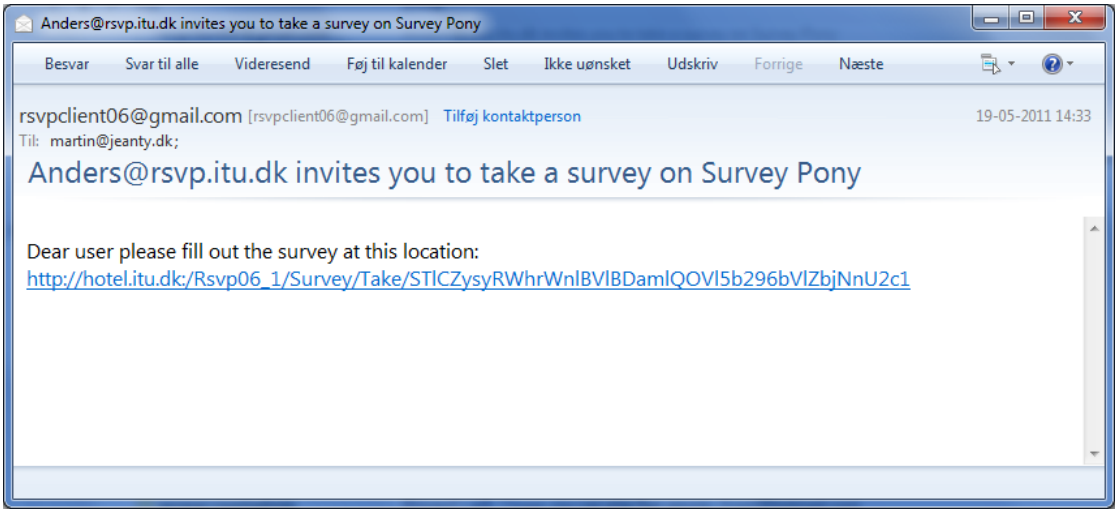
4. The participant has been added to the survey and you get a confirmation.



5. If you go back to manage participants the participant appears in the list and you can now send an invitation by email by clicking the "Send Email" link to send an invitation to a specific participant or use the "Send out survey invitations" link which will send out invitations to all participants in your survey.



6. The Email the system generates to participants has a unique link so that others can't simply try and guess the server link and answer server, an example follows.

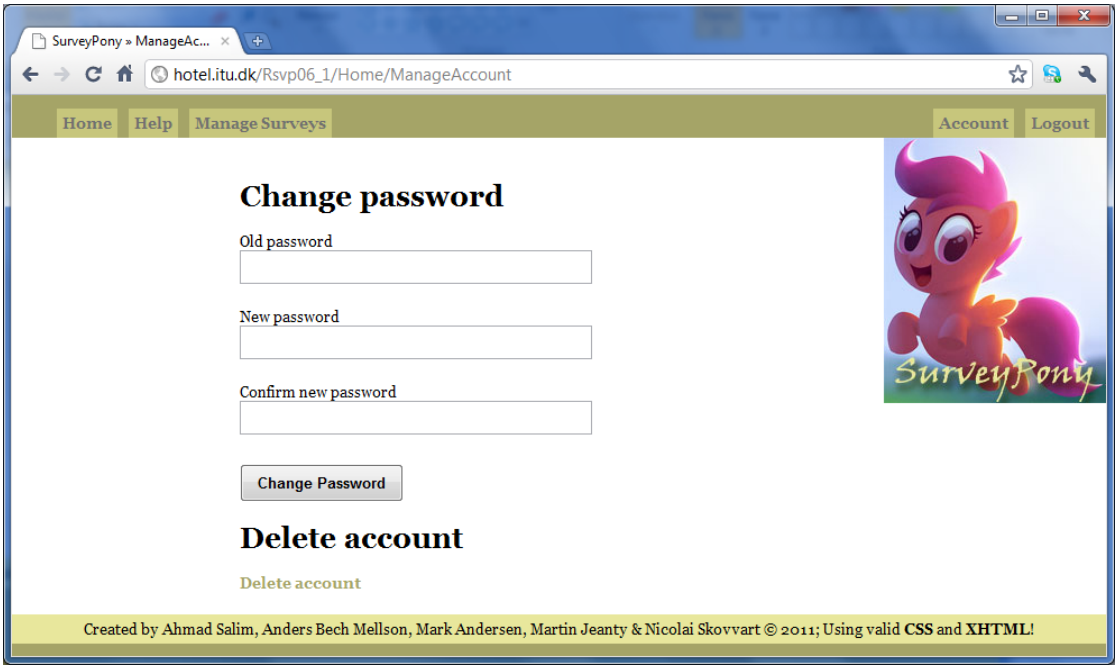


5.6 Account

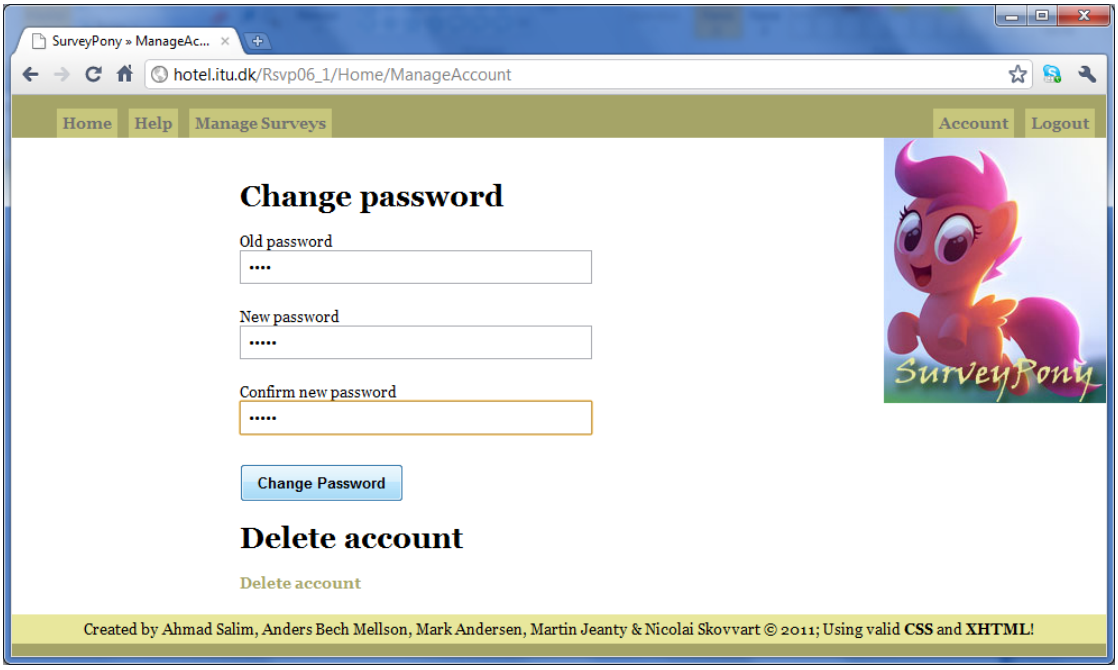
This manual explain how to change the password for your account.

1. In the account management you can change your password or delete your account

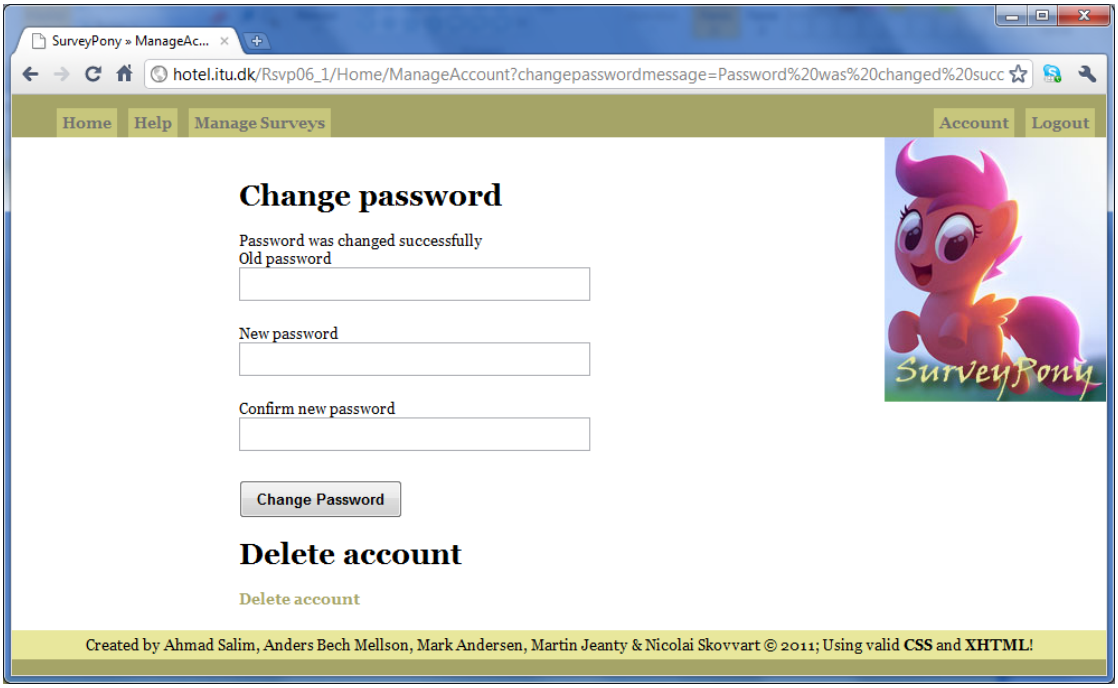
Note: The delete account will delete your account with no further notice.



2. Change password: Enter your current password and the new password twice, finish by clicking "Change Password".



3. Given that the supplied current password was correct and the new password and confirmation matched your password has now been changed.



Chapter 6

Technical Description

6.1 Server

The purpose of the server is to handle data. Thus, it must support methods for supporting insertion and extraction of data. The server should be able to display, create and delete surveys as well as fetching the results of a survey. It should also allow people to answer surveys. It should also allow listing, adding and removing participants from a survey.

The web services of the server is implemented in F# using the ADO.NET entity framework for object-relational mapping. The actual services are exposed in C# using ASP.NET web services. The reason for using ASP.NET web services is because of issues with exposing web services natively from F#.

All services are structured hierarchically in modules, but there is a "super module" WebService.fs, that gathers all methods for the C# class to expose via the ASP.NET web service.

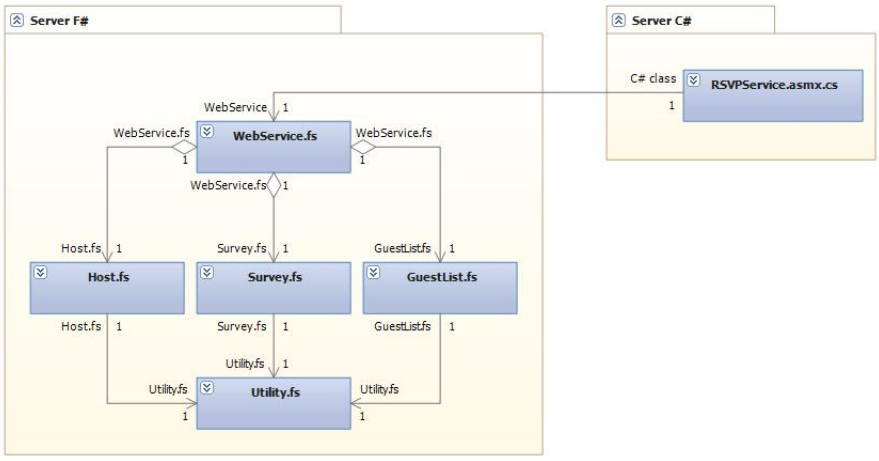


Figure 6.1: Illustration of the structure of the server.

The reasoning behind the module structure was to logically separate methods and to improve distributed work.

Host.fs contains various methods to create, delete and alter host accounts. Survey.fs contains

methods to create, delete and view surveys as well as some other methods, for example view statistics for a survey. It also contains the module Invitee that contains methods to view, add and delete participants from a survey and the module Answers that contains methods to submit answers, remove answers as well as completion of the survey. It uses this approach for clients that allow partial answers to surveys to allow completion later. We did not, however, have time to add a way to get the partial answers returned from the service. We also have a Utility-module with help-methods that the various modules are utilizing such as password-hashing, email-validation, user-validation and so forth.

Survey.fs is also utilizing the provided Template-parser that exists in a seperate F# project. The parser has been slightly modified and also supports the question-type "Matrix" that allows for grid-questions. We did not, however, have time to implement this in the client. The parser now also interprets underscores in the survey-name as spaces on request of the Singapore-team.

6.1.1 Data insertion

Insertions is generally just querying the database and requiring proper authorization.

Authorization is accomplished by requiring a Host-struct containing a host-email and password. First a check is done to ensure that the email/password-combination is correct, then that it has access to perform the requested method. For example, a host must own a survey to be able to delete it. An alternative to this would be to set up sessions with the clients, but the group decided that we would rather have the server stateless to support better scaling.

6.1.2 Data extraction

While data extraction also involves querying the database, it can be a bit more tricky returning the result to the client. For simple web services such as AuthenticateHost, that return a boolean, it is easy to return the result directly. Other methods return collections, some of which are simple enough that you can just return arrays of values or arrays of simple structs containing values.

Some of the more tricky methods such as displaying all the data required to show the survey or to show the results of people who have answered the survey require some extra thought. The result of the web service must include enough information for the client to be able to invoke other web services, such as the IDs of questions and options. We decided to implement it using XML as it's a very flexible format that can be used in just about any programming language. It does require some effort to generate the XML and for clients to parse and interpret it. It could potentially have been implemented using complex structs as well.

6.2 Client

6.2.1 Architecture

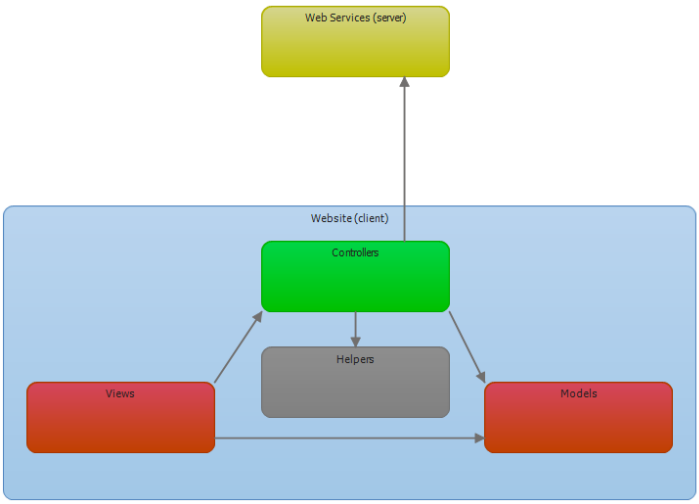


Figure 6.2: The architecture of our client.

As we have chosen the ASP.NET MVC framework to program our client in, it seemed natural to code our client according to the Model-View-Controller architecture pattern.
The model-view-controller architecture pattern is a pattern for separating the main concerns in an application, manipulating data using business logic (the model), presenting the application through an interface (the view) and managing user input and executing requests by communicating to the two layers (the controller).

This pattern has multiple advantages 1) it enables readability and maintainability of code, as changing business logic only happens one place, and adding another UI would not change the base logic 2) it enables better testability because of the independence of the three modules, thus making testing logic and presentation easier and finding errors faster and 3) lastly it allows independent development of user interface and business logic which allows easier distribution of work items
To achieve the optimal usage of the framework we had tried to adhere as strictly as possible to this pattern, but chose also to refactor some of the most used functionality in the controllers into helper classes for more reusability. Furthermore we focused on making the controllers the only part that communicated with the web services (which also present some kind of bussiness logic), to ensure the maintainability of our client.

6.2.2 Authentication and authorization

By design authentication is needed in our web services, requiring user email and password on every important web service call. We thus had to find a suitable way of storing these data, when a user gets authenticated as to not ask the user his email and password every time. As the ASP.NET MVC framework already had a way of basic authentication called "Forms Authentication", which already took care of creation of encrypted cookies and handling authorization

on web pages (by using the authorize attribute), we wanted to extend the capabilities of the framework, as to reuse as much as possible.

The specific problem with just using forms authentication, was that one could only save a users name, and we needed to also store the user password for use on the web services. A way this could be resolved was proposed by Microsoft[1], and the idea is to change the default user and user.identity objects in the controllers with custom implementations using their respective IPrincipal and IIdentity interfaces, and then modifying the cookie to include this new type of information.

We thus created our own implementation of User called RSVPPrincipal containing our custom RSVPIdentity that included the data for email and password. To store the new information in a cookie, we thought it would be easier to create the default forms authentication cookie and then modify the information afterwards (in postauthenticate request).

In this way we could use the default way of routing unauthorized requests (with the authorize attribute on actions that required authentication), and also store the information for easy access in the controllers by just casting the user.identity object to an RSVPIdentity and getting the custom attributes from there.

6.2.3 Survey Management

A key feature of a survey system is of course survey management. In this part there were many problems here to be solved:

1. How to format the XML given by the service for seeing and taking the survey
2. How to list and delete the surveys that a user has
3. How to manage participants for a given survey
4. How to create a survey in an easy intuitive way

Displaying a survey

For displaying the surveys to the user for taking we chose a fairly simple approach. A user gains access to the survey by using an encoded URL (containing the survey id and his email address), and then got presented with the corresponding survey given by the survey id (viewing the survey does not require authentication). The user is presented with a HTML web form interpreted from the given XML-formatted survey data. The form was converted to the corresponding input types by iterating through the XML by using XPath expressions with the help of the LINQ-to-XML framework. When answers are given the given form data is parsed accordingly to the type of question and a call is made to the question submission web services. If there were any errors (e.g. the user has not answered the required questions), the user is redirected back to the survey with an explanation of what was done wrong.

[1] `http://www.asp.net/security/tutorials/forms-authentication-configuration-and-advanced-topics-cs`, step 5

Listing the surveys

Listing the surveys was implemented through a web page that called the list survey web service for the currently authenticated user. For convenience we also used the list page to show links for deletion, participant management and viewing statistics for each survey. The deletion part was simply implemented as a call to an action in the survey controller that takes a survey id, and then delegates the call to the DeleteSurvey web service.

Participant handling

Participant management was done through a web page where one could add and delete participants (via a specific controller), and the user also had the ability in this page to email a participant or all participants with a generated link for the survey. For sending emails the WebMail helper in the ASP.Net framework was used, where one just input one's credentials, the email server and the message to be sent; and it automatically would handle all SMTP communication.

Displaying answers and charts

A more interesting part of the client was showing the results as charts. In this problem there was three parts to be solved: 1) extracting all the important information from the XML-formatted answers 2) transferring the data for presentation and 3) presenting the data in beautiful charts. For extracting the data we used LINQ extracting question types and description for each question, then if the type was "text" we chose to print all answers in a list. If the data was of type "checkboxes" or "select" all options and corresponding vote count was extracted and saved in some variables to be sent for chart generation. For generating charts a controller that utilized the ASP.NET charting API was created. This charting API contains methods for generating various types of charts using data series consisting of a list of x-values which was used to display the option text, and a corresponding list of y-values which was used to display the vote count for that given option. After generating the chart we saved the image to a memory stream, and returned a temporary file for display into the statistics HTML page.
Although it might seem an easy task to transfer the data from the view to the chart generation controller, there was no easy way to transfer lists using the default RouteValue dictionary for generating url parameters (which is mapped to function parameters by the framework).
The default value for the framework was just to convert the list to a string (using the default ToString method), and the given input when it was a list of strings that was needed was *"System.Collection.Generic.List"*, the default ToString implementation of the .NET framework. A solution was found, which required referring each item by the parameter list name and then an index referring to their position(e.g. the first item of the xs list should be given as xs[0]). By formatting the list elements this way the URL parameters was given correctly, and the chart generation worked as expected.

Survey creation

The most interesting part of our client was the survey creation feature. For survey creation we wanted a more user friendly solution than just plain editing of **template** which required some knowledge and technical skills to use.
We decided to use an HTML-based form solution, which resulted in some interesting issues. The

first issue was that we did not know how many questions a user would create in advance and furthermore how many options there would be for each question, thus we could not create a plain static HTML page. We thus had to use client side Javascript, to dynamically alter the HTML DOM[2], and adding and removing questions could be done dynamically. Although there are some drawbacks with this method. The HTML was written as strings in the Javascript which made it harder to maintain and test. The other drawback was that if there was some erroneous input the user would be redirected back and only the questions written in the static part are remembered, thus forcing the user to retype his questions.
After taking the user input, we had to convert the form data to the **template** language. While we used a notation in the form keys to explain what the type is (e.g. q[0] for the first question, and q[0]o[0] for the first option for the first question), we still had to identify them using our source code. We came up with a solution of using regular expressions to identify a form key depending on the notation we used, and then we had to solve how to add information for a question. Our solution was to create an intermediate type storing the information in there, and using a sorted map with an integer (identifying the ordering of the question) as key, and the intermediate question type as values. For easier conversion to template we made each question be able to convert itself dependent on its type. The only thing that was left was to assemble the full template with a name, and call the web service to create the survey.

[2]The Document Object Model - `http://www.w3.org/DOM/`

Chapter 7

Test strategy

To ensure that our applications work as intended, we have created a series of tests, to try and test out the RSVP web services on the server, and our implementation of the RSVP client. For the server part we have chosen to only do unit testing of the most important methods, as many methods are called independently of others and integration should be achieved if every tested method work. For the client part we have chosen a model-view-controller pattern, and thus we are testing each important part of the three layers (model, view, controller).

7.1 Server

7.1.1 Unit test

For the unit testing of the server part of our program, we have chosen to use the black box testing technique on the web service interfaces.

The reason we have chosen to do black box testing over white box testing is that even though we have access to all our code, many parts of our code is LINQ queries of which we do not know how they will be compiled.n The other part is, that the branches that we do have in our queries are only for defensive programming and thus a good black box test will cover those as well. Lastly it is because we needed to write a lot of valid data ourselves to test the many methods in different ways, we could not use some kind of automated white box testing and manual white box testing would have taken too much time compared to the weight of testing.

While we have had some helper functions in the server in addition to the web service methods, we have focused on testing the web service methods only. The main reason is that the helper functions are small and not of great importance, in addition to that they will be tested inside the web services.

In our testing of the server, we have tried to cover as many use cases as possible, doing both negative tests (with invalid, non-existing and duplicate data) and positive tests. We have also tried to isolate many of the test methods to include one service method only, but sometimes it was not possible (e.g. SubmitQuestionAnswer is needed to SubmitSurveyAnswers). Each of our test classes were structured in such way that they did an interaction with the real database, but in transactions (which were rolled back in the end of each test), as to get some kind of realism not achievable with a simulated database.

After writing our tests we uncovered 17 flaws in the web services, mostly in the email checking and null reference input, which we had to fix. The testing of the server thus resulted in a greater confidence that our web services work as expected and that the SMU group should be able to

rely on our services without any problems. For further information about our server tests please refer to our appendix H.1: server testing plan.

7.2 Client

7.2.1 Unit test

For the unit testing of our client wad chosen to programmatically blackbox test the models and helper-classes in our architecture. The reason we are not directly unit testing our controllers is because all of the most important functionality was refactored into helper classes (for maintainability and usability), and the majority of the functionality left was calls to the web service (which was tested in the server part) and passing the data to the clients (which we rely on the framework upon).
The reason we had not unit tested the views programmatically was first because of the usage of client-side scripting (with javascript), which would make it harder to test in C# without simulating a browser, and second because it would take a lot of time mocking the controllers and framework and then also the user clicks. In anyway the controllers and the views will be covered by the integration tests.

In the testing of the models in our application, we tried to test the model-constraints we had put on for client-side validation. As we do not know how the constraints are implemented by the framework, and wanted to test that we did have enough constraints and that the constraints we have were valid, the only real option was to use extensive black-box testing.
The way we tested the models was simple, as we mocked a controller created the model with the variables we wanted to test and then passed the model to the controller. Afterwards it was just an option to check if the *ModelState* (a validation state that the controller has for checking model integrity) was valid.

As with right to the helpers, it would seem that we could have used white-box testing as we had written all of them ourselves. While some of the more complicated methods could be white-box tested, we chose to emphasize testing functionality with black-box tests as they would be faster to write and also enables the mentality to test for special input (like null values), we might have missed checking in our branches. It should be noted that we did not test very simple helper methods, as many of the only gave readability and maintainability for some of the code, as opposed to much functionality.

In the end unit testing our clients models and helpers made us uncover some of the hidden errors in our client application, proved that many important functions of our client application works and increased the reliability in those modules. For the full testing schemes please look at appendix H.2 : client unit testing plan.

7.2.2 Integration testing

While the unit tests of some of the most important layers in our client's architecture resulted in increased reliability in the client, we had not proved that our modules worked well with each other and there were still some parts that was not completely covered by our unit tests i.e. our controllers and our views.

While most of the real functionality on the controllers was in the helpers, we thought that it would not be a complete guarantee that our controllers worked as expected. As such there could still be some error checks in which the controllers behaved erroneously, or some views that missed data from the given controllers. We had also used the framework option of annotating methods and classes that required authorization with the authorize attribute, and testing that it worked the way we wanted would have been hard using unit tests, and thus the only resort was to uncover errors by integration testing.

Testing the views with the use of integration test was clearly a good idea for us as it also was easier to test our client-side scripts. Alternatively the client-side scripts should have been tested with a javascript testing framework, which required much time and in which we were inexperienced with. Using the views is also one of the only ways to do an external integration test, as it is the only things that takes external user interaction. External testing of the view, also reflects the way a normal user would use our client, and thus would give us great confidence to put in a production environment if there were not many failures.

The way we chose to integration test our client, was to identify key scenarios in our clients functionality e.g. creating a user, and trying out various possibilities, both for positive tests and negative tests. In the example I had mentioned some of the things that could be tested is creating an user with an invalid email, or trying to create a user with an email that already exist. In this way we systematically tried to uncover some of the most ordinary scenarios an ordinary user could run into.

Overall while our integration test did not cover 100% of all possible scenarios — which is usually impossible — we are confident that our systematic approach to testing uncovered many of the most important flaws, and thus we can conclude that we have a good and reliable client. For the full testing schemes please refer to appendix H.3 : client integration testing plan.

Chapter 8

Documentation of communication

As part of our project we had to do distributed development together with a team in Singapore. This task provided some challenges regarding communication, tools for collaborative work, cultural differences, language barriers and our different level of technical expertise. The technical experience along with communication ended up being the biggest challenges, more on that later. In order for our group to be well prepared for the collaboration, we first had to handle our internal communication.

8.1 Internal communication

Alongside this project we had a course in system development and project organization (BSUP). Here we were taught several different ways to do project management and software development. These models all have a different perspective on how to best handle either project management or software development. Models, we have seen, for project management include PRINCE2[1], IPMA[2], Scrum[3] and more. Descriptions of each model is excluded here, but we have provided links in the foot notes for the interested reader to learn more about each model. Our work together with Singapore (SMU) started before we were finished with the BSUP course. This meant that we hadn't learned all the models when we started organizing ourselves. Because of this we constructed our own model, because the ones we had seen thus far werent fit for this project (in our point of view).

The model we came up with was quite basic. Every day we started out with a group meeting containing discussion about work progress, and ideas that possibly needed to be converted into work assignments. We also ended each day with a similar meeting, our status meeting. Early on we identified some roles to distribute among ourselves. We wrote all the roles we identified down and handed each of them out to group members. We had quite many roles at the start, but as the process advanced we found that some were dispensable.

We kept roles such as responsible persons in each are of development - server, client and test. And we also kept the role of project manager and had a person responsible for documentation.

After we were taught about Scrum in BSUP, we realized that our own model was actually quite close to that development model. The main difference being that we had split the group in different roles and that is not normal practice in Scrum development.

[1] `http://www.prince-officialsite.com/`
[2] `http://www.ipma.ch/`
[3] `http://www.scrumalliance.org/`

8.2 External communication

Our first meeting with our partner group from SMU was arranged for us by the professors in the course (Niels and Ben). We got a time slot for a video meeting at ITUs video conference room and the SMU group was in a conference room at their university. Before our first meeting we sent an email to our new partners explaining who we were and specifying our contact information. Along with these formalities we also sent a message saying that we were looking forward to be working together. This email was answered with contact information from their side, which was nice to have before the first meeting. They also acknowledged our greeting, and replied with their own. Everything was well underway for a great start. At our first meeting we got a chance to see each other and put a face on the names from the emails. We also got a first impression on the way our partners were handling their internal organization. At this meeting we agreed on a weekly time slot for meetings. And we also agreed upon which communication tools we would use initially. Read the entire meeting log in appendix C on page 51.

8.2.1 Agreements from our initial meeting

Weekly Meeting Time slot
Thursdays
1630 - 1830 (Singapore)
0930 - 1130 (Copenhagen)

Email
itu@bechmellson.com - sends email to all ITU students in the project.
singapore@bechmellson.com - sends email to all SMU students in the project.

Dropbox
We setup a shared folder for exchanging files.

Google Docs
A shared folder was opened to have easy realtime collaboration on documents.

Skype
We agreed to do video meetings Skype.

All meeting logs are attached as appendix C on page 51.

8.3 Reflection on the process of the international collaboration

8.3.1 Experiences with used techniques

Our experiences with the technology we used for collaboration was overall very satisfying. Google Docs proved to be an indispensable tool for collaborative writing in realtime. At most meetings we would have the document in question open on computers both at ITU and SMU. So that while we were discussing its contents, we could (re)write as we spoke. This was a very effective way of understanding each other quickly and without fuzz. Since both groups could see our agreements being written down in realtime, this eradicated many errors. We found that when we saw things

in writing it was very easy to understand each other, and correct misunderstandings as they occurred instead of having to iron out the communication errors later.

We also had an overall good experience with Dropbox for sharing files. However we did experience some smaller problems with the way we used it. We had a shared folder for files, which we at ITU thought would be for non-document type files. Now we realize that we didnt agree on which type of files to put in Dropbox, this was an error from our team. This did cause some confusion, because where would we find the document we were looking for, at Google docs or in Dropbox? Our group at ITU tended to leave written documents to Google docs and only have drawings and odd file types on Dropbox. Where as the SMU group placed both written documents in Dropbox and on Google Docs. This is something that we will be more clear on in the future. Basically we forgot to communicate what was in our heads, we assumed that SMU thought the same. This is a valuable lesson we will bring forward with us.

Our first video meeting with SMU went great, primarily because we had access to ITUs video conferring system. The quality of both video and audio was very good when using this system. When we later tried to do our weekly meetings in Skype however, the story was quite different. We learned that in order to do group video chat in Skype, you need to have a pro account. Something which we didnt have. So instead we tried to gather around one computer at each university in order to overcome this obstacle. But it didnt really work that great with the built-in cameras in our computers. We were simply to many persons in front of the computer. The camera couldnt fit us all in its frame. So what we learned from this was that either we should get Skype pro accounts (or similar software). Or we should have a more professional web camera. What we ended up doing instead was to drop the video chat. Instead we used a combination of audio and text chat. Which actually worked very well. As with the collaborative writing in Google docs, it was nice to have text chat along with the audio. This helped us break down language barriers. Often just the pronunciation of words led to misunderstanding between our teams. What struck us though after the first meetings was how much communication is lost when talking via video, audio or text chat. All the finer details of communication, such as facial expressions, tonality of the voice etc, all these communication skills seemed impossible to benefit from with the tools we used. It makes you realize how much you actually communicate with a person standing besides you, without even talking.

Finally we used email for communication outside of the weekly meeting time slot. We setup two email groups for ease of use. We created an email group called singapore@bechmellson.com, that would forward to all the members at SMU. Similarly we setup a group called itu@bechmellson.com which would send to all members at ITU. This proved to be a handy way of communicating because one would only need to remember two email addresses.

All the technologies described so far has been our collaboration tools with the students from Singapore University. Internally we used some of the same technologies and some additional ones as well.

Our primary internal project tool has been Google Docs. Here we have written our daily meeting logs (see appendix C on page 51), reports and all of our technical documents. This has been very effective and its only had upsides for our usage. We also had a shared folder on Dropbox which was only for the ITU team, which had documents from various programs that Google docs doesnt support. Mainly model design in OmniGraffle[4] and diagrams in OmniPlanner[5]. Later in our project, after we had the lesson on Scrum in system development and project organization (BSUP), we started using a new tool called Pivotal Tracker[6]. Tracker is an online collaborative tool for Scrum development. Since we identified our own project management style as being so

[4]Program for drawing data models, work breakdown structures and much more.

[5]Tool for keeping an overview on project management.

[6]`http://www.pivotaltracker.com/`

close to Scrum, we decided to switch to regular Scrum for the remainder of the project. Tracker supports daily meetings, where we share ideas that can be broken down into tasks. These tasks are then assigned to a member of the group. Each development cycle is called a sprint and we decided to use a length of one week for every cycle. Tracker seems to be very capable of handling the way we develop and work together.

8.3.2 How did we perform as a group

Before we started working on any design of our server, we sat down as a group and formed a group constitution. In this document we agreed on an overall structure of our group. We identified and assigned individual roles, work times and discussed sanctions if people were late. This document (See appendix F on page 89) has not been used much throughout the process, but it formed the basis of what our group was. And it has helped form a good working morale. Throughout the process our group has been very focused, professional and has kept our eyes on the target. The overall rating of the groups performance is very satisfying. Every piece of code has been tested, rewritten and finalized into what we think is beautiful and clean code. Even in the harder parts of the collaboration where SMU wanted to change the code into something we thought was worse, our group prevailed and came up with solutions that would benefit both parties.

As a group we have had a tremendous boost because we were lucky to have an expert programmer in the group. Ahmad was very quick to pick up the F# language and has programmed the base of our code. Besides being an excellent programmer he also showed great skills in helping the rest of the group get up to speed with the coding and F# skills. He excelled at pair programming.

We didnt experience big technical issues while working. We think it was because we had tested all the technologies before entering the collaboration with SMU. We did have some minor glitches in our collaboration with SMU. This was caused by the difference in our technical experience. This led to some miscommunication about what our web services should be named and what functionality they should provide. But we quickly overcame these problems and found solutions that benefited both parties.

8.3.3 Experiences with SMU collaboration

We have been pleased with our collaboration with SMU. The majority of their group seemed very nice and was forthcoming. As a whole we think it was a great collaboration. It was however not without problems. As briefly mentioned above, the difference in coding experience between the teams was the biggest technical problem in the collaboration. We at ITU are used to talking about code and its functionality. Whereas it seemed that not everyone at SMU had the same level of code knowledge. This quickly became a problem when we needed to do the web services. An example was that SMU wanted a boolean returned every time a call was successful, whereas we thought it would be better if they only got an exception if the call failed. We ended up doing both, since they insisted on having the boolean returned.

Our biggest communication problem with SMU came about because of the way they had structured their group internally. We never got any confirmation of how they handled their internal group on paper, so this next part is just how we think it was handled based on our working experience with them. They had appointed one guy as their leader, which handled all communication and decision making. This was a big problem, firstly because of the bottleneck effect. Secondly because the appointed person didnt seem to have a lot of technical experience. This led to many confusing discussions, as we were talking together - but not understanding

each other. We could have a meeting with him, agree on something. Then when we later talked about it again, he couldnt understand what we were talking about.

Another problem was that they didnt seem to read through all the needed documents. An example was that they told us, in the beginning of the project, that is wasnt mandatory for them to use the TEMPLATE language. They said they would rather use XML, which was okay with us, as long as they had got an approval from their professor, which they assured us they had. We asked to see their mandatory requirements, just to clarify since we were told otherwise by our professor. In that document it stated they had to use TEMPLATE. We explained this to them, and then they werent so sure that their professor approved on using XML instead. This behavior caused quite some problems throughout the project, as they didnt always read the documents we used for communicating with them either. Often their project leader would ask us to do things, we already had done and documented.

The most heated discussion with them came as we were about to finalize our agreements on the web services. We had a shared document with the description of all the services (See appendix G on page 93). And we wanted to agree on the contents before coding. It was very hard to finish this document, because they were very slow in reading it and giving their feedback. When they finally approved it, they had their technical person approve it. A couple of days after that, their leader wanted to change a lot in the services again. This was quite annoying, since the changes were not very relevant, and because they had already approved it. But his mind was set and we had to implement these changes in order for the collaboration not to break down.

What went well though was the general communication with the rest of their group. Especially the communication with their technical person was very good. He was always polite, effective and constructive. There were also two other persons in their group, but we never really talked to these two directly.

Next time we are going to do international collaboration, it is essential to break down the decision structure beforehand. It made absolutely no sense, that their leader should overrule great decisions their team had already made. But since its an internal thing for each team to decide, its probably not something we have any influence on.

Some of the positive things we can take into our next ventures are things like written agreements, email forwarding and our general use of collaborative tools.

Chapter 9

Recommendations

In this chapter we have identified a couple of things that we think could have been better. We think these could improve a future revision of the system. The recommendations was something we could have done, if we had more time to work on the project.

9.1 System improvements

Usability testing was something we didn't manage to fit in our schedule for the client development. Usability testing could have have improved the user experience of the program. The problem is that we think as programmers, because we are. And since we have written the program ourselves we are blind to the usability problems the client surely has.

The user experience of the client could also be vastly improved if we worked more on the interface design. This is a natural extension of usability testing, where to place buttons, which areas to highlight and so on.

Unified login would also be an improvement, both to the server and the client. We could support OpenID or Google login which prevents the user from having too many logins to remember.

Mobile applications that could access both the server and the client. A client application that would allow users to take a survey on their mobile phones seems like something that could benefit the system. And if administrators could create surveys and see statistics on their mobile devices, this would surely be an added value.

9.2 Code improvements

In relation to code improvements there are three main areas that mainly could be looked at for improvement: 1) *features*, 2) *functionality* and 3) *security*.

9.2.1 Features

While our data model is rich enough to support many types of question types and formatting, in the end our client only supported the four given types (”Select” , ”Check Boxes” , ”Explanation” and ”Text”).
If we had more time to improve our functionality we would first complete the implementation of the matrix type of questions. This is because we already had extended the **template** parser to support it and only needed to output it as such in the XML in the web services, and then of

course implement the client support. In addition there are also a lot of small features we have not extended the **template** parser to support, but which our datamodel supported (including some of which our client already supports) :

- Support for multiple pages
- Support for other types of questions
- Support for "other" answer types in the select and radiobutton question types
- Support for specifying input as numerical
- Support for minimum and maximum amount of answers on specific questions
- Support for one-to-one mapping between questions and options in matrices (i.e. one can not select the same option twice for two questions in a matrix)
- Support for start and expiry dates of surveys

In the long run one feature we really wanted to do was to support advanced statistical computations from our client. This includes filtering results for a question according to answers given in an other question, statistical accuracy, and the ability to approximate statistical distributions for a given set of answers e.g. the normal distribution (with average and variance).

9.2.2 Functionality

Functionality wise there is some things we wanted to improve in our client, but in the urge of time was excluded.
The first functionality in our client which could be improved is survey creation. In survey creation we wanted to implement a live preview function, so that when one had finished writing a question, the user would have a better idea on how it would be displayed to the client, and maybe choose a better question type. Another thing that could be improved was an easier way to edit surveys, as now one has to delete the old one and then create a new survey and add the participants again, which could be easily automated "behind the scenes".

9.2.3 Security

In the end there could also be small improvements of security in our application.
The way our applications work now is that the web services need the user password send in clear-text to the server when authenticating, and then a hashing of the password would be done. This is obviously not very secure, and only gets worse on the client side as the password has to be saved someway. The way we have saved the password on the client side is to make an encrypted cookie and store it in there, but this is against any common security practice as it would be an vulnerability if the encryption algorithm is defeated and someone gets hold of the cookie file. A small but better improvement is to include the hashing algorithm client side, and only send and store the hash. This of course is still a way of authenticating the user if someone decrypts the cookie, but be more protective of the user password if used anywhere else. Something that is much better than that is generating an authentication token that is time-limited when a user is authenticated, and thus minimize the flow of user credentials. A small note to be taken is that we in a production environment would use HTTPS for web services that needed authentication but was omitted during development for ease of debugging, and thus there would be no direct clear text transmission of user credentials.

Chapter 10

Conclusion

We have had a great experience working on the project. The course has been well prepared and executed with precision. The parts we enjoyed most have been F# and the learning experience from working together with a team on the other side of the world. The collaboration process had its difficulties, but we have learned from this. And in the future we are now more prepared to work in an international team.

The video meetings arranged by the professors worked very well, we think that more scheduled meetings in the conference rooms would benefit the process.

We would like to have seen SMU's requirements document when we started working together. We think it would be nice to have all official requirements on the wiki.

We also had some technical issues with the server provided from ITU. A working server throughout the project would improve the process.

In spite of minor technical glitches and collaboration issues we have had a great experience. Our overall conclusion is that the project was a success. We have learned a lot and are grateful of that.

Appendix A

Review report

RSVP06s review of RSVP02s draft.

Requirements specification

- Fine introduction that explains why requirements are important
- Vaguely specified business goal - the only requirement is that usability must be considered what about requirements related to survey results?
- Stakeholders the text specifies these but there is no relation to specific requirements which takes each group into consideration. E.g. its important that representatives of the employees are included in the usability test
- Tasks gives a fine picture of what the system must support but the text is mixing some things together in the manage questionnaire-section there is both the creation of questionnaires and responding to them, they should probably be split up in different tasks
- Generally the structure of this section is good the text gets around the purpose, goal, stakeholders and functional-/non-functional requirements

data model

- It is good that there is a picture at the start of the data model section as it provides an overview of the data model
- The data model is also so descriptive that one has the feeling that one knows all about it. I would go as far as saying it is overly descriptive
- Many tables share the same similar attributes (as ID), that share functionality, and repeated descriptions are thus not necessary
- Also some reflections are tautological (e.g. on page 9 section 2.1.1 line 2-3). This statement (and the like) somehow leaves the impression that the text is underestimating the readers abilities

- Despite these unfortunate wordings the reasoning in the text seems very good, and it has some valid arguments for the things written
- We would suggest that instead of doing a description the way it is written now it should focus on more generally describing each relation, and only highlighting special and non self-explanatory attributes

Reflection

- Fine introduction, simple and outlines what the reader is about to read
- The text seems to have the SMU vs general collaboration parts reversed? It says that there wasnt any problems, right after that it starts describing the problems that were
- Great section in the text about the cultural differences that had an impact on the final specifications of the web services
- The part about womens rights seems a bit unclear to me, why is it in this document. Did the group experience problems related to this? Also the section is asserting a lot without background, I see that there is a footnote - is it better documented in the text it is quoted from?
- Good finishing section about the strategic problems. Maybe it would be nice to read about how the group actually handled this part

Appendix B

Explanation of data model

Here is an explanation of each entity and attribute in the data model.

- **GuestList**; *A list containing guests who can access a specific survey*
  - Id; *Unique identifying number*
  - Name; *Name for the guest list*
- **Guest**; *Entity used to dissolve a many to many connection. Guest is a user in a guest list Since a user can be in many guest lists, here the guest entity allows for a user to have a different name in each guest list*
  - Name; *Name for the guest*
- **User**; *This entity is a placeholder for all the information we need on a user*
  - Id; *Unique identifying number*
  - Email; *Users email*
  - Password; *Needed if the user is an admin, otherwise it can be NULL*
  - Name; *Users name, chosen by the user*
- **Answer**; *Answer to a question*
  - Text; *Text string if the type of question allows for the user to input text*
  - IsComplete; *Boolean to indicate if the question is completed*
- **Role**; *The different roles a user can have, admin, participant and so on*
  - **Id**; *Unique identifying number*
  - **Name**; *Name of the role*
- **Participant**; *Dissolves a many to many connection. A user can be a participant in many surveys and have many roles*
- **Survey**; *The actual survey that can be created and taken by users*
  - **Id**; *Unique identifying number*
  - **Title**; *Title of the survey*

  - **Description**; *Short description of what the survey is about*
  - **StartDate**; *When is the survey accepting answers from*
  - **EndDate**; *When is the survey accepting answers to*

- **Group**; *Placeholder for questions*
  - **Id**; *Unique identifying number*
  - **Title**; *Title of the group*
  - **Page**; *Which page is this group supposed to be on*
  - **Ordering**; *Ordering is used to order groups on the same page. The lower the ordering number, the closer the group will be to the top of the page*
  - **IsCascading**; *Used to force only one answer per row in a matrix question*
  - **IsMatrix**; *Is this group a matrix question*

- **Question**; *Question that a user participating in a survey can answer*
  - **Id**; *Unique identifying number*
  - **Ordering**; *Ordering is used to order questions on the same page. The lower the ordering number, the closer the question will be to the top of the page*
  - **Text**; *Text describing the question*
  - **MinAnswers**; *Is this a required question? It is if MinAnswer is above 0*
  - **MaxAnswers**; *Maximum number of answers accepted for this question*
  - **IsNumeric**; *Is the question a numeric type, e.g. what is your age*

- **Type**; *Which type is this question*
  - **Id**; *Unique identifying number*
  - **Name**; *Name of the question type*

- **Option**; *Subtype of question. Can contain more text or an "other" field*
  - **Id**; *Unique identifying number*
  - **IsOther**; *Is this an "other" field?*

Appendix C

Meeting Logs and agendas

On the next pages you will find all our meeting- logs and agendas from the SMU collaboration. We have excluded a longer chat log from the 22.03.2011 but it can be found online see this footnote[1].

[1] `http://cl.ly/6ohL` Chat log from the 22.03.2011

- **Agenda for 24.03.11(Video Conference)**
    - Tests, how will SMU test our webservice methods so we're sure everything works as intended and that we are in agreement on the method's behavior.(E.g. sample code, we will provide sample code for each webservice method however). Reporting of any bugs etc.
    - Is it ok to use attributes in XML, does everybody know how this works?

- **Agenda for 21.03.11**
    - Finalize various XML contracts such as the response from creating surveys, and the format in which statistics will be presented
- **Agreements for 21.03.11**
    - ITU will make sure that authentication and registration is working on the webservice for SMU to test
    - ITU will finalize renaming web-service-methods to use common-terms
    - ITU will add an update-password webservice-method
    - ITU will use exceptions where needed
    - SMU will create suggestions for survey-id-xml and statistics-xml

- **Agenda for 17.03.11**
    - statistics (send images or raw data? XML?)
    - show survey
        - we need a draft on the XML data (maybe as C# objects???)
    - deadline for SMU?
        - which date, the 24/3 or the 4/4
    - Web services corrected walkthrough
- **Agreements from 17.03.11**
- ITU will change void methods to return booleans to indicate that processing has occured
- Common terms:

Survey- Collection of Questions to form a questionnaire
Question- Set of words to get an Answer
Answer- Set of words to respond to a Question
Results- Set of Answers to the Questions in a Survey
Host- User who creates Surveys
Invitee- User who gives Answers to the Questions in a Survey
Guest List- Collection of Invitees who can access a Survey

- Suggestions for chat
    - easy example for iterating over users
- ITU implements authenticate web-service method
- SMU will consider the client handling email
- ITU will make an example on how to use webservices without parsing xml
- SMU will make a sample of answer structure (probably xml)

- **Agenda for 10.03.11**
    - Agreement on data communication

        - document here: http://goo.gl/qLJIa
    - Which web service methods should be available to SMU
    - Statistics - is it okay with SMU if we send pure survey data?
        - (no generated images and such)
        - XML
    - Are there any other issues SMU needs ITU to take care of in order to be able to continue with their work?
- **Agreements from 10.03.11**
    - SMU sends XML outline structure to ITU.
    - Struct explanation, ITU will send sample code on how to do a user struct.
        - and a link to Microsoft struct explanation
        - This is done and placed in dropbox\Documentation\struct_example.txt
    - Along with the template, the autentication information(e-mail/password) should also be sent
    - Uncertainty about the Template language – ITU checks with Niels what the requirements are on both ends.
    - SMU looks at the webservice list and gives possible input
    - Once we agree upon the webservice we will put the documentation(wsdl) on the wiki page
    - We use Google docs for meeting agendas and agreements in the future

First meeting between the ITU group and the SMU group.

We agreed on using dropbox for sharing documents and meeting summaries – we will create the shared folder and make sure that invitations are sent out.

We agreed on reoccurring meetings every thursday at 9:30 (ITU time)/16:30(SMU time).

ITU Project

Date of Meeting:	March 3, 2011
Venue of Meeting:	Online conference
Time of Meeting:	1630 hours
Minutes Prepared By:	Lim Cheening

1. Purpose of Meeting

- Introduction
- Decide on preferred forms of communication
- Establish weekly meeting session

2. Attendance at Meeting

Name	*Attendance*
Alvin Chua	
Lim Cheening	
Kang Kai Xin	
Gabriel Yee	
Nicolai	
Martin	
Ahmad	
Mark	
Anders	

3. Meeting Notes, Decisions, Issues

Scheduled Weekly Meeting Time slot

- Thursday
  - Time: 1630 - 1830 (Singapore)
  - Time: 0930 - 1130 (Copenhagen)

Communication Tools

- eMails
- Skype
- Dropbox
- Google doc

4. Next Meeting					
Target Date:	*March 10, 2010 (Thursday)*	*Time:*	*1630 hours (Singapore)*	*Location:*	*Video Conference*
Objectives:	• Set project deadlines • Clarify requirement specification • Sharing on project expectations and contribution				

ITU Project

Date of Meeting:	March 10, 2011
Venue of Meeting:	Online conference
Time of Meeting:	1630 hours
Minutes Prepared By:	Lim Cheening

1. Purpose of Meeting

- Set project deadlines
- Clarify requirement specification
- Sharing on project expectations and contribution

2. Attendance at Meeting

Name	*Attendance*
Alvin Chua	
Lim Cheening	
Kang Kai Xin	
Gabriel Yee	
Nicolai	Absent
Martin	
Ahmad	
Mark	Absent
Anders	

3. Meeting Notes, Decisions, Issues

Agreement on data communication

- document here: http://goo.gl/qLJIa
- Use of Google docs for meeting agendas and agreements in the future

Project Matters

- SMU sends XML outline structure to ITU
- Struct explanation, ITU will send sample code on how to do a user struct
    - And a link to Microsoft struct explanation
    - This is done and placed in *dropbox\Documentation\struct_example.txt*

- Along with the template, the authentication information(e-mail/password) should also be sent
- SMU looks at the webservice list and gives possible input
- Once ITU agree upon the webservice, ITU will put the documentation(wsdl) on the wiki page

Uncertainty

- Template language
    - ITU checks with Niels what the requirements are on both ends

4. Next Meeting					
Target Date:	*March 17, 2010 (Thursday)*	*Time:*	*1630 hours (Singapore)*	*Location:*	*Video Conference*
Objectives:	•				TBA

ITU Project

Date of Meeting:	March 17, 2011
Venue of Meeting:	Online conference
Time of Meeting:	1630 hours
Minutes Prepared By:	Lim Chee Ning

1. Purpose of Meeting

- Set project deadlines
- Clarify requirement specification
- Deadline for SMU
- Web services corrected walkthrough

2. Attendance at Meeting

Name	*Attendance*
Alvin Chua	
Lim Cheening	
Kang Kai Xin	
Gabriel Yee	
Nicolai	
Martin	
Ahmad	
Mark	
Anders	

3. Meeting Notes, Decisions, Issues

Project Matters

- ITU will change void methods to return booleans to indicate that processing has occurred

Common terms:

- Survey- Collection of Questions to form a questionnaire
- Question- Set of words to get an Answer
- Answer- Set of words to respond to a Question
- Results- Set of Answers to the Questions in a Survey
- Host- User who creates Surveys

- Invitee- User who gives Answers to the Questions in a Survey
- Guest List- Collection of Invitees who can access a Survey

Suggestions for chat

- easy example for iterating over users
- ITU implements authenticate web-service method
- SMU will consider the client handling email
- ITU will make an example on how to use webservices without parsing xml
- SMU will make a sample of answer structure (probably xml)

4. Next Meeting					
Target Date:	*March 21, 2010 (Monday)*	*Time:*	*TBA*	*Location:*	*Video Conference*
Objectives:	• TBA				

ITU Project

Date of Meeting:	March 21, 2011
Venue of Meeting:	Online conference
Time of Meeting:	1600 hours
Minutes Prepared By:	Lim Chee Ning

1. Purpose of Meeting

- Project updates
- Web services request

2. Attendance at Meeting

Name	*Attendance*
Alvin Chua	Absent with Reason
Lim Cheening	
Kang Kai Xin	Absent with Reason
Gabriel Yee	
Nicolai	
Martin	
Ahmad	
Mark	
Anders	

3. Meeting Notes, Decisions, Issues

Common understanding:

- Host & invitees are the same...
- host can be invitee/ invitee can be host
- invitee not need password
- on create survey, system will check user for password (see if they have account)
- else system will prompt user to create account
- after creating acc, will auto login, and user able to create survey

- ITU

- ITU will make sure that authentication and registration is working on the webservice for SMU to test (by March 21, 2011)
- ITU will finalize renaming web-service-methods to use common-terms
- ITU will add an update-password webservice-method
- ITU will use exceptions where needed

- SMU
  - SMU will create suggestions for survey-id-xml and statistics-xml

For detail information, please check out google doc- "Web Services Corrected 15.03.11"

4. Next Meeting					
Target Date:	*March 24, 2010 (Thursday)*	*Time:*	*1630 (Singapore)*	*Location: SMU*	*Video Conference*
Objectives:	• TBA				

ITU Project

Date of Meeting:	March 24, 2011
Venue of Meeting:	Online conference
Time of Meeting:	1600 hours
Minutes Prepared By:	Lim Chee Ning

1. Purpose of Meeting

- Project updates
- System Testing

2. Attendance at Meeting

Name	*Attendance*
Alvin Chua	
Lim Cheening	
Kang Kai Xin	
Gabriel Yee	
Nicolai	
Martin	
Ahmad	
Mark	
Anders	

3. Meeting Notes, Decisions, Issues

All web services will be up by today (24th March 2011)

System Testing

- There is no web services for "Update Survey"
- How SMU will go about testing ITU web service methods
  - ITU to add dummy data
- Reporting of any bugs

Other project matters

- Is it ok to use attributes in XML, does everybody know how this works?
- SSD in progress

4. Next Meeting					
Target Date:	*March 31, 2010 (Thursday)*	*Time:*	*1630 (Singapore)*	*Location: SMU*	*Video Conference*
Objectives:	•				TBA

Appendix D

F# Assignments

D.1 Ahmad

```
   //Exercise Material
module List =
    let rec foldr f b xs =
        match xs with
        | [] -> b
        | x::xs -> f(x, foldr f b xs)
    let rec foldl f b xs =
        match xs with
        | [] -> b
        | x::xs -> foldl f (f(x,b)) xs

type 'a BinTree =
  | Leaf
  | Node of 'a * 'a BinTree * 'a BinTree

//Exercise #1
let sqr x = x + x

//Exercise #2
let pow x n = System.Math.Pow x,n
let pow2 (x:int) (n:int) = int <| System.Math.Pow (float x, float n)
let rec pow3 x n = x*(pow3 x (n-1))

//Exercise #3
let dup (str:string) = str + str
(*There is now need for explicit type annotation on ^ because it is made for
    string concatenation only,
    where + can be used by int, float etc. Although ^ in F# is a legacy construct
        for ML compatability*)

//Exercise #4
let rec dupn (str:string) n = str + dupn str (n-1)

//Exercise #5
//If t1 is less than t2 then t1 is assumed to be one day later
//Not validating input
let timediff t1 t2 =
    let diff = ((fst t1)-(fst t2))*60+(snd t1)-(snd t2)
    if diff > 0 then diff else 1440 + diff
```

```
//Exercise #6
let minutes t = timediff t (0,0)

//Exercise #7
let rec downTo n = if n >= 1 then n::(downTo (n-1)) else []
let rec downTo2 n =
    match n with
    | 0 -> []
    | n -> n::(downTo (n-1))

//Exercise #8
let rec removeEven (l:int list) =
    match l with
    | e1::e2::t -> e1::(removeEven t)
    | e::[] -> e::[]
    | _ -> []

//Exercise #9
let rec combinePair (l:int list) =
    match l with
    | e1::e2::t -> (e1,e2)::(combinePair t)
    | _ -> []

//Exercise #10
let explode (str:string) = str.ToCharArray() |> Array.toList
let rec explode2 (str:string) = if str.Length > 0 then (str.[0])::explode2(str.
    Remove(0,1)) else []

//Exercise #11
let implode (l: char list) = l |> List.foldr (fun (el,acc) -> el.ToString() + acc)
     ""
let implode2 (l: char list) = List.foldBack (fun el acc -> el.ToString() + acc) l
    ""
let implodeRev (l: char list) = l |> List.foldl (fun (el,acc) -> el.ToString() +
    acc) ""
let implodeRev2 (l: char list) = List.fold (fun acc el -> el.ToString() + acc) ""
    l

//Exercise #12
let toUpper str = explode str |> List.map(System.Char.ToUpper) |> implode
let toUpper1 str = (explode >> List.map(System.Char.ToUpper) >> implode) str
let toUpper2 str = str |> (implode << List.map(System.Char.ToUpper) << explode)

//Exercise #13
let palindrome (str:string) = (str.ToLower()) = (implodeRev << explode) (str.
    ToLower())

//Exercise #14
//No input check
let rec ack (m,n) =
    match (m,n) with
    | (0,n) -> n+1
    | (m,0) -> ack(m-1,1)
    | (m,n) -> ack(m-1,ack(m,n-1))

//Exercise #15
let time f =
    let start = System.DateTime.Now
    let res = f()
    let finish = System.DateTime.Now
    (res, start, finish)
```

```
let timeArg1 f a =
    let (res, start, finish) = time (fun () -> f(a))
    res,(finish-start)

//Exercise #16
let rec inOrder tree =
    match tree with
    | Leaf -> []
    | Node(v, l, r) ->
        (inOrder l) @ v::(inOrder r)

//Exercise #17
let rec mapInOrder f tree =
     match tree with
     | Leaf -> Leaf
     | Node(v, l, r) ->
        let lv = (mapInOrder f l)
        Node(f(v), lv, mapInOrder f r)

(*If you use a function like the following: fun x -> printf "%A" x; x+x;
 then mapInOrder and mapPostOrder will return the same tree, but will print the
     values in different order*)

//Excercise #18
let foldInOrder f acc tree = tree |> inOrder |> List.foldl f acc

//Exercise #19-22

type expr =
  | Const of int
  | If of expr * expr * expr
  | Bind of string * expr * expr
  | Var of string
  | Prim of string * expr * expr

let rec lookup l key =
 match l with
 | (k,v)::t -> if k = key then v else lookup t key
 | [] -> failwith("Unknown variable name:" + key)

let rec eval expr env =
  match expr with
    | Const i -> i
    | If(exprb, expr1, expr2) -> if (eval exprb env) <> 0 then (eval expr1 env)
       else (eval expr2 env)
    | Bind(varname, expr1, expr2) -> eval expr2 ((varname, eval expr1 env)::env)
    | Var(varname) -> lookup env varname
    | Prim("-", expr1, expr2) -> eval expr1 env - eval expr2 env
    | Prim("+", expr1, expr2) -> eval expr1 env + eval expr2 env
    | Prim("max", expr1, expr2) -> System.Math.Max(eval expr1 env, eval expr2 env)
    | Prim("min", expr1, expr2) -> System.Math.Min(eval expr1 env, eval expr2 env)
    | Prim("=", expr1, expr2) -> if (eval expr1 env) = (eval expr2 env) then 1
       else 0
    | Prim(opr, _, _) -> (printfn "Operation %s not supported" opr; 0);

//Examples
let ``2+2;`` = Prim("+", Const 2, Const 2);
let ``x=10;y=5;x+y;`` = Bind("x", Const 10, Bind("y", Const 5, Prim("+", Var "x",
    Var "y")))
let ``Overshadow example: x=10;y=5;x=3;x+y;`` =  Bind("x", Const 10, Bind("y",
    Const 5, Bind("x", Const 3,Prim("+", Var "x", Var "y"))))
let ``Swap example: x=10;y=5; z=x; x=y; y=z; y==(x+x);`` =
```

```
Bind("x", Const 10,
  Bind("y", Const 5,
    Bind("z", Var "x",
     Bind("x", Var "y",
      Bind("y", Var "z",
        Prim("=", Var "y",
          Prim("+", Var "x", Var "x")))))))
let ``If example: if max(10,2) == 10 then 1 else -1`` =
If(Prim("=",Prim("max", Const 10, Const 2), Const 10),
  Const 1,
  Const -1)
let ``Unknow variable example: x;`` = Var "x"
```

D.2 Mark

```
   // F# Assignments by Mark Philip Andersen mpan@itu.dk
open System

//Exercise 1
let sqr x = x*x;;

//Exercise 2
let pow x n = Math.Pow(x,n);;
let pow2 (x: int) (n:int ) = int(Math.Pow((float)x,(float)n));;
let rec pow3 (x:int) (n:int) =
    if n = 0 then 1
    else x*pow3 x (n-1);;

//Exercise 3
let dup (a:string) = a+a;;
// + will assume a number, ^ will assume a string.

//Exercise 4
let rec dupn (a:string) (n:int) =
    if n=1 then a
    else a+dupn a (n-1);;

//Exercise 5
let timediff (hour:int, minut:int) (hour_2:int , minut_2:int) =
    (hour_2-hour)*60 + (minut_2 - minut);;

//Exercise 6
let minutes (hour:int, minut:int) =
    timediff (00,00) (hour,minut);;

//Exercise 7
let rec downTo (n:int) =
    if n = 1 then 1::[]
    else n::downTo (n-1);;
let rec downTo2 (n:int) =
    match n with
    | 1-> 1::[]
    | n-> n::downTo2 (n-1);;

//Exercise 8
let rec removeEven (mylist:int list) =
  match mylist.Length with
  | 0 -> []
  | 1 -> mylist
  | 2 -> mylist.Head::[]
  | _ -> mylist.Head::removeEven(mylist.Tail.Tail);;
```

```
//Exercise 9
let rec combinePair (mylist:int list) =
  match mylist.Length with
  | 0 -> []
  | 1 -> []
  | _ -> (mylist.Head,mylist.Tail.Head)::combinePair(mylist.Tail.Tail);;

//Exercise 10
let explode (a:string) = List.ofArray (a.ToCharArray());;
let rec explode2 (a:string) =
  match a.Length with
  | 1 -> a.Chars(0)::[]
  | _ -> a.Chars(0)::explode2(a.Remove(0,1));;

//Exercise 11
let rec foldr f b xs =
  match xs with
    [] -> b
  | x::xs -> f(x, foldr f b xs);;
let implode (charlist:char list) = charlist |> foldr (fun (char,a) -> char.
    ToString() + a) "" ;;
let implode2 (charlist:char list) = charlist |> List.fold (fun a char -> a+char.
    ToString()) "" ;;
let implodeRev (charlist:char list) = charlist |> foldr (fun (char,a) -> a+char.
    ToString()) "" ;;
let implodeRev2 (charlist:char list) = charlist |> List.fold (fun a char -> char.
    ToString()+a) "" ;;

//Exercise 12
let toUpper (datstring:string) = implode(List.map (fun a -> System.Char.ToUpper(a)
    )(explode datstring));;
let toUpper1 (datstring:string) = datstring |> (explode >> List.map (fun a ->
    System.Char.ToUpper(a)) >> implode);;
let toUpper2 (datstring:string)= datstring |> explode |> (implode << List.map (fun
     a -> System.Char.ToUpper(a)));;

//Exercise 13
let palindrome (datstring:string) =
    if datstring = implodeRev (explode(datstring)) then true
    else false;;

//Exercise 14
let rec ack (m,n) =
  match m, n with
  | _ when m = 0 -> n+1
  | _ when m > 0 && n = 0 -> ack(m-1,1)
  | _ -> ack(m-1,ack(m,n-1));;
//ack (3,11);;
//val it : int = 16381

//Exercise 15
let timeArg1 f a =
  let start = System.DateTime.Now in
  let res = f (a) in
  let finish = System.DateTime.Now in (res, finish - start);;

//Exercise 16
type 'a BinTree =
    Leaf
  | Node of 'a * 'a BinTree * 'a BinTree;;
```

```
let rec inOrder btree =
  match btree with
    Leaf -> []
  | Node(n, lefttree, righttree) -> inOrder(lefttree) @ n :: inOrder(righttree);;

//For test
let IBT = Node(43, Node(25, Node(56,Leaf, Leaf), Leaf), Node(562, Leaf, Node(78,
    Leaf, Leaf)));;

//Exercise 17
let rec mapInOrder f tree =
  match tree with
  | Leaf -> Leaf
  | Node(n, lefttree, righttree) ->
    let left = mapInOrder f lefttree
    Node(f(n),left,mapInOrder f righttree);;

//Exercise 18
let foldInOrder f acc t = List.fold f acc (inOrder t);;

//Exercise 19 through 22 combined
type expr =
  | Const of int
  | If of expr * expr* expr
  | Prim of string * expr * expr
  | Bind of string * expr * expr
  | Var of string

let rec lookup env y =
  match env with
  | [] -> failwith ("Unknown variable: "+y)
  | (x,v)::rest -> if x=y then v else lookup rest y

let rec eval env expr =
  match expr with
    Const i -> i
  | Prim("-", expr1, expr2) -> eval env expr1 - eval env expr2
  | Prim("+", expr1, expr2) -> eval env expr1 + eval env expr2
  | Prim("max", expr1, expr2) -> System.Math.Min(eval env expr1, eval env expr2)
  | Prim("min", expr1, expr2) -> System.Math.Max(eval env expr1, eval env expr2)
  | Prim("=", expr1, expr2) -> if eval env expr1 = eval env expr2 then 1 else 0
  | If(expr1,expr2,expr3) -> if eval env expr1 <> 0 then eval env expr2 else eval
      env expr3
  | Var a -> lookup env a
  | Bind (var, expr1, expr2) -> eval ((var,eval env expr1)::env) expr2
  | Prim(opr, _, _) -> (printfn "Operation %s not supported" opr; 0);

//Some examples
// 100+100
eval [] (Prim("+", Const(100), Const (100)));;
// 100-100
eval [] (Prim("-", Const(100), Const (100)));;
// 100 = 100;
eval [] (Prim("=", Prim("max",Const(1),Const(100)), Prim("min",Const(100), Const
    (1000))));;
// if (100 = 100) then 9000 else 8999
eval [] (If(Prim("=", Prim("max",Const(1),Const(100)), Prim("min",Const(1000),
    Const(100))),Const(9000),Const(8999)));;
//a=100, b=100, a+b
eval [] (Bind("a",Const(100),Bind("b",Const(100),Prim("+",Var "a", Var "b"))));;
```

D.3 Martin

```
   // @Author: Martin Jeanty Larsen, mjla@itu.dk

// Exercise 1
let sqrt x = x * x
let x = sqrt 2

// Exercise 2

// A

let pow (x:float, n:float) = System.Math.Pow(x, n)
let y = pow (2.0, 4.0)

// B

let pow2 (x:int, n:int) = int(System.Math.Pow((float)x, (float)n))
let y2 = pow2 (2,4)

// C

let rec pow3 (x:int, n:int) =
    match n with
        0 -> 1
        | n -> x * pow3(x, n-1)
let y3 = pow3 (2, 5)

// Exercise 3

let dup s = s+s
let y4 = dup("Hey")

// Exercise 4

let rec dupn (s:string, n:int) =
    if n = 1 then s
    else s + dupn(s,n-1)
let y5 = dupn("Test", 4)

// Exercise 5

let timediff (h1,m1)(h2,m2) = System.Math.Abs((h1 * 60 + m1) - (h2 * 60 + m2))
let y6 = timediff(14,30)(17,40)

// Exercise 6

let minutes (x,y) = timediff(x,y)(0,0)
let y7 = minutes(1,0)

// Exercise 7

let rec downTo n =
    if n = 1 then [n]
    else n::(downTo(n-1))
let y8 = downTo 5

let rec downTo2 n =
    match n with
        1 -> [n]
        | _ -> n::(downTo2(n-1))
```

```
let y9 = downTo 5

// Exercise 8

let rec removeEven xs: list<int> =
    match xs with
    [] -> []
    | [_] -> xs
    | g::xs -> g::removeEven(xs.Tail)
let y10 = removeEven[1;2;3;4;5]

// Exercise 9

let rec combinePair (xs: list<int>) =
    match xs with
    [] | [_] -> []
    | g::xs -> (g,xs.Head)::combinePair(xs.Tail)
let y11 = combinePair[1;2;3;4]
let y12 = combinePair[1;2;3;4;5]
let y13 = combinePair[]
let y14 = combinePair[1]

// Exercise 10

// A
let explode (s: string) = List.ofArray(s.ToCharArray())
let y15 = explode("star")

// B

let rec explode2 (s: string) =
    if s.Length = 0 then [] else
    let i = s.Chars(0)
    let q = s.Remove(0,1)
    i::explode2(q)
let y16 = explode2("star")

// Exercise 11

let rec foldr f b xs =
    match xs with
    | [] -> b
    | x::xs -> f(x, foldr f b xs)

let implode (xs: char list) = xs |> foldr (fun (el,acc) -> el.ToString() + acc) ""

let y17 = implode(['a';'b';'c'])

// B

let implode2 (xs: list<char>) =
    List.foldBack (fun el remaining -> el.ToString() + remaining) xs ""

let y18 = implode2(['a';'b';'c'])

// C

let rec foldl f b xs =
    match xs with
    [] -> b
    | x::xs -> foldl f (f(x,b)) xs
```

```
let implodeRev (xs: list<char>) =
    xs |> foldl (fun (el,remaining) -> el.ToString() + remaining) ""

let y19 = implodeRev(['a';'b';'c'])

// D

let implodeRev2 (xs: list<char>) =
    List.fold (fun remaining el -> el.ToString() + remaining) "" xs

let y20 = implodeRev2(['a';'b';'c'])

// Exercise 12

// A
let toUpper (str : string) =
    explode(str) |> List.map(System.Char.ToUpper) |> implode

let y21 = toUpper("test")

// B
let toUpper1 (str : string) =
    (explode >> List.map(System.Char.ToUpper) >> implode) str

let y22 = toUpper1("test")

// C
let toUpper2 (str : string) =
    str |> (implode << List.map(System.Char.ToUpper) << explode)

let y23 = toUpper2("test")

// Exercise 13

let palindrome (str : string) =
    (str.ToLower()) =
        (implodeRev << explode) (str.ToLower())

let y24  = palindrome("Anna") // true
let y25  = palindrome("Ann") // false

// Exercise 14

let rec ack (x,y) =
    match (x,y) with
    | (0,n) -> y+1
    | (x,0) -> ack(x-1,1)
    | (x,y) -> ack(x-1,ack(x,y-1))

let y26 = ack(3,11)

// Exercise 15

// A
let time f =
    let start = System.DateTime.Now
    let res = f()
    let finish = System.DateTime.Now
    (res, start, finish)

let y27 = time (fun () -> ack (3,11))
```

```
// B
let timeArg1 f a =
    let (res, start, finish) = time (fun () -> f(a))
    res,(finish-start)

// Exercise 16

type 'a BinTree =
  | Leaf
  | Node of 'a * 'a BinTree * 'a BinTree

let rec inOrder tree =
    match tree with
    | Leaf -> []
    | Node(v, l, r) ->
        (inOrder l) @ v::(inOrder r)

// Exercise 17

let rec mapInOrder f tree =
     match tree with
     | Leaf -> Leaf
     | Node(v, l, r) ->
        let lv = (mapInOrder f l)
        Node(f(v), lv, mapInOrder f r)

// Exercise 18

let foldInOrder f remaining tree =
    tree |> inOrder |> foldl f remaining

// Exercise 19

type expr =
  Const of int
  | Prim of string * expr * expr

let rec eval expr =
  match expr with
    Const i -> i
  | Prim("-", expr1, expr2) -> eval expr1 - eval expr2
  | Prim("+", expr1, expr2) -> eval expr1 + eval expr2
  | Prim("=", expr1, expr2) -> if eval expr1 = eval expr2 then 1 else 0
  | Prim("min", expr1, expr2) -> System.Math.Min(eval expr1, eval expr2)
  | Prim("max", expr1, expr2) -> System.Math.Max(eval expr1, eval expr2)
  | Prim(opr, _, _) -> (printfn "The operation %s is not supported" opr; 0)

// Exercise 20

let y28 = eval(Prim ("=", Const 32, Const 51))
let y29 = eval(Prim ("=", Const 51, Const 51))
let y30 = eval(Const(34))

// Exercise 21

type expr =
    | Const of int
    | If of expr * expr * expr
    | Prim of string * expr * expr;;

let rec eval expr =
  match expr with
```

```
    Const i -> i
  | Prim("-", expr1, expr2) -> eval expr1 - eval expr2
  | Prim("+", expr1, expr2) -> eval expr1 + eval expr2
  | Prim("=", expr1, expr2) -> if eval expr1 = eval expr2 then 1 else 0
  | Prim("min", expr1, expr2) -> System.Math.Min(eval expr1, eval expr2)
  | Prim("max", expr1, expr2) -> System.Math.Max(eval expr1, eval expr2)
  | Prim(opr, _, _) -> (printfn "Operation %s not supported" opr; 0)
  | If(expr1,expr2,expr3) -> if eval expr1 <> 0 then eval expr2 else eval expr3;;

// Exercise 22

type expr =
  Const of int
  | If of expr * expr * expr
  | Bind of string * expr * expr
  | Var of string
  | Prim of string * expr * expr;;

let rec lookup env y =
  match env with
  | [] -> failwith ("Unknown variable: " + y)
  | (x,v) :: rest -> if x=y then v else lookup rest y;;

let rec eval expr env =
  match expr with
    Const i -> i
  | Prim("-", expr1, expr2) -> eval expr1 env - eval expr2 env
  | Prim("+", expr1, expr2) -> eval expr1 env + eval expr2 env
  | Prim("=", expr1, expr2) -> if eval expr1 env = eval expr2 env then 1 else 0
  | Prim("min", expr1, expr2) -> System.Math.Min(eval expr1 env, eval expr2 env)
  | Prim("max", expr1, expr2) -> System.Math.Max(eval expr1 env, eval expr2 env)
  | Prim(opr, _, _) -> (printfn "Operation %s not supported" opr; 0)
  | If(expr1,expr2,expr3) -> if eval expr1 env <> 0 then eval expr2 env else eval
      expr3 env
  | Bind(string,expr1,expr2) -> eval expr2 <| (string,eval expr1 env)::env
  | Var(string) -> lookup env string
```

D.4 Nicolai

```
   //Nicolai Skovvart

//Exercise 1: Write a function sqr : int -> int so that sqr x returns x^2
let sqr x =
    x*x

//Exercise 2: Write a function pow : float -> float -> float so that pow x n
    returns
//x^n. You can use the library function: System.Math.Pow. Now write an integer
    version, pow2
//: int -> int -> int. For pow2 you have to write types explicitly and use type
    cast on
//System.Math.Pow. Now write pow3 : int -> int -> int using recursion.
let pow basenumber exponent =
    basenumber**exponent

let pow2 (basenumber:int) (exponent:int) =
    (int)((float)basenumber**(float)exponent)

let rec pow3 number exponent =
    match exponent with
    | 1 -> number
```

```
    | _ -> number * pow3 number (exponent-1)

printfn "%f" (pow 3.0 5.0)
printfn "%i" (pow2 2 3)
printfn "%i" (pow3 3 4)

//Exercise 3: Write a function dup : string -> string that concatenates a string
    with itself.
//If you are using +, then you can write the type of the argument explicitly
    not necessary if you use
//^. Why is that?
//Answer: Because otherwise the default type is assumed to be an integer. ^ only
    works for strings, so there's no ambiguity, but it is not part of the standard
     F# configuration
let dup (string:string) =
    string + string
printfn "%s" (dup "hippopotomonstrosesquipedaliophobia")

//Exercise 4: Write a function dupn : string -> int -> string so that dupn s n
//creates the concatenation of n copies of s.
let rec dupn (string:string) copies =
    match copies with
    | 1 -> string
    | _ -> string + dupn string (copies-1)
printfn "%s" (dupn "Tora" 3)

//Exercise 5: Assume the time of day is represented as a pair (hh, mm) : int * int
    . Write a
//function timediff : int * int -> int * int -> int so that timediff t1 t2
//computes the difference in minutes between t1 and t2.
let timediff (hh, mm) (hh2, mm2) =
    (hh2-hh)*60 + mm2-mm
printfn "%i" (timediff (12, 45)(13, 59))

//Exercise 6: Write a function minutes : int * int -> int to compute the number of
//minutes since midnight. Easily done using the function timediff.
let timefrommidnight (hh, mm) =
    timediff (0, 0)(hh, mm)
printfn "%i" (timefrommidnight (5, 23))

//Exercise 7: Write a function downTo : int -> int list so that downTo n returns
    the nelement list [n; n-1;    ; 1].
//    Use if-then-else expressions to define the function.
//    Define the downTo2 function being equal to downTo except that you now must
    use pattern
//matching.
let rec downTo int =
    if(int=1) then 1::[]
    else int::downTo (int-1)

let rec downTo2 int =
    match int with
    | 1 -> 1::[]
    | _ -> int::downTo2 (int-1)
printfn "%A" (downTo 5)
printfn "%A" (downTo2 9)

//Exercise 8: Write a function removeEven : int list -> int list so that
//removeEven xs removes the even-indexed elements from the list xs:
//    removeEven [x1; x2; x3; x4; x5;    ] = [x1; x3; x5;    ]
//    removeEven [] = []
//    removeEven [x1] = [x1]
```

```
let rec removeEven list =
    match list with
    | x1::x2::xs -> x1::removeEven xs
    | x -> x

printfn "%A" (removeEven [1;2;3;4;5;6;7;8;9])

//Exercise 9: Write a function combinePair : int list -> (int*int) list so that
//combinePair xs returns the list with elements from xs combined into pairs. If xs
     contains an
//odd number of elements, then the last element is thrown away:
//     combinePair [x1; x2; x3; x4] = [(x1,x2);(x3,x4)]
//     combinePair [x1; x2; x3] = [(x1,x2)]
//     combinePair [] = []
//     combinePair [x1] = []
//Hint: Try use pattern matching.
let rec combinePair (list:int list) =
    match list with
    | x1::x2::xs -> (x1, x2)::combinePair xs
    | _ -> []

printfn "%A" (combinePair [1; 2; 3; 4; 5])

//Exercise 10: Write a function explode : string -> char list so that explode s
//returns the list of characters in s:
//explode   s t a r   = [   s   ;   t   ;   a   ;   r   ]
//Hint: if s is a string then s.ToCharArray() returns an array of characters. You
    can then use
//List.ofArray to turn it into a list of characters.
//Now write a function explode2 : string     char list similar to explode except
    that
//you now have to use the string functions s.Chars (or .[]) and s.Remove, where s
    is a string.
//The definition of explode2 will be recursive.
let explode (string:string) =
    List.ofArray<char> (string.ToCharArray())

let rec explode2 (s:string) =
    match s.Length with
    | 1 -> [s.[0]]
    | _ -> s.Chars 0::explode2(s.Remove(0, 1))

printfn "%A" (explode "Batman")
printfn "%A" (explode2 "Robin")

//Exercise 11: Write a function implode : char list -> string so that implode s
//returns the characters concatenated into a string:
//implode [   a   ;   b   ;   c   ] =   a b c
//Hint: Use foldr from lecture.
//Now write a function implode2 : char list     string similar to implode. Now you
//just have to use the build in List operation List.foldBack. The functions foldr
    and
//foldBack are similar except for their type, hence the way arguments are given.
//Now write a function implodeRev : char list     string so that implodeRev s
//returns the characters concatenated in reverse order into a string:
//implodeRev ['a';'b';'c'] =   c b a
//Hint: Use foldl from lecture.
//Now write a function implodeRev2: char list     string similar to implodeRev.
    You
//have to use the build in List operation List.fold.
let implode (list: char list) =
    new System.String(list |> Array.ofList)
```

```
let implode2 (list:char list) =
    List.foldBack(fun char act -> (string)char+act) list ""

let rec implodeRev (list: char list) =
    match list.Length with
    | 1 -> (string)list.[0]
    | _ -> implodeRev list.Tail + (string)list.[0]

let implodeRev2 (list: char list) =
    List.fold(fun char (act:char) -> (string)act+char) "" list

let captain = ['C';'A';'P';'T';'A';'I';'N']
let planet = ['P';'L';'A';'N';'E';'T']
printfn "%A" (implode captain)
printfn "%A" (implode2 planet)
printfn "%A" (implodeRev captain)
printfn "%A" (implodeRev2 planet)

//Exercise 12: Write a function toUpper : string -> string so that toUpper s
    returns
//the string s with all characters in upper case:
//toUpper    H e j    =    H E J
//Hint: Use System.Char.ToUpper to convert characters to upper case. You can do it
     in one line
//using implode, List.map and explode.
//Aside: Write the same function toUpper1 using forward function composition
//((f >> g) x = g(f x)).
//Aside: Write the same function toUpper2 using the pipe-forward operator (|>) and
     backward
//function composition (<<).
let toUpper string =
    implode(List.map(fun c -> System.Char.ToUpper(c))(explode string))

let toUpper1 =
    explode >> List.map(fun c -> System.Char.ToUpper(c)) >> implode

let toUpper2 string =
    string
    |> explode
    |> (implode << List.map(fun c -> System.Char.ToUpper(c)))


printfn "%A" (toUpper "caps␣lock")
printfn "%A" (toUpper1 "equals␣cruisecontrol")
printfn "%A" (toUpper2 "for␣cool")

//Exercise 13: Write a function palindrome : string -> bool, so that palindrome s
//returns true if the string s is a palindrome; otherwise false. A string is
    called a palindrome if it
//is identical to the reversed string, eg,    A n n a    is a palindrome but
      A n n    is not.
let palindrome string =
    if(toUpper string=(implodeRev (explode (toUpper string)))) then true
    else false

let palindrome2 str =
    let origStr = toUpper str
    match (str |> toUpper |> explode |> implodeRev) with
    | o when o = origStr -> true
    | _ -> false
```

```
printfn "%A" (palindrome "Anna")
printfn "%A" (palindrome "Michael Bolton")
printfn "%A" (palindrome2 "Anna")
printfn "%A" (palindrome2 "Michael Bolton")

//Exercise 14: The Ackermann function
//is a recursive function where both value and number of
//mutually recursive calls grow rapidly.
//Write the function ack: int * int -> int that implements the Ackermann function
    using
//pattern matching on the cases of (m,n) as given to the right above. What is the
    result of
//ack(3,11).
//Answer: result is 16381
let rec ack (m, n) =
    match (m, n) with
    | (0u, _) -> n + 1u
    | (_, 0u)-> ack(m-1u, 1u)
    | (_, _) -> ack(m-1u, ack(m, n-1u))

printfn "%A" (ack(3u, 11u))

//Exercise 15: The function time f : (unit ->    a ) ->    a * TimeSpan below times
     the
//computation of f x and returns the result and the real time used for the
    computation.
//let time f =
//   let start = System.DateTime.Now in
//   let res = f () in
//   let finish = System.DateTime.Now in
//   (res, finish - start);
//Try compute time (fun () -> ack (3,11)).
//Write a new function timeArg1 f a : (   a ->    b ) ->    a ->    b * TimeSpan
    that
//times the computation of evaluating the function f with argument a. Try timeArg1
     ack
//(3,11).
//Hint: You can use the function time above if hide f a in a lambda (function).
let time f =
  let start = System.DateTime.Now
  let res = f ()
  let finish = System.DateTime.Now
  (res, finish - start)

printfn "%A" (time(fun() -> ack(3u,11u)))

let timeArg1 f a =
  time(fun()-> f a)

printfn "%A" (timeArg1 ack (3u, 11u))

//Exercise 16: Write a function inOrder :    a BinTree ->    a list that makes an
    in-order
//traversal of the tree and collect the elements in a result list.
//Hint: See slide for definition of in-order.
type 'a BinTree =
    Leaf
  | Node of 'a * 'a BinTree * 'a BinTree

let rec inOrder binTree =
    match binTree with
    | Leaf -> []
```

```
    | Node(n, left, right) -> inOrder left @ n :: inOrder right

let intBinTree = Node(43, Node(25, Node(56,Leaf, Leaf), Leaf), Node(562, Leaf,
    Node(78, Leaf, Leaf)))

printfn "%A" (inOrder intBinTree)

//Exercise 17: Write a function mapInOrder : ( a -> b) -> a BinTree ->
    b
//BinTree that makes an in-order traversal of the binary tree and apply the
    function on all nodes in
//the tree. Can you give an example of why mapInOrder might give a result
    different from
//mapPostOrder, but the result tree retruned in both cases is still the same.
let rec mapInOrder f tree =
    match tree with
    | Leaf -> Leaf
    | Node(n, left, right) ->
    let leftSide = mapInOrder f left
    Node(f n, leftSide, mapInOrder f right)

printfn "%A" (mapInOrder (fun n-> printfn "%A" n; n + 2) intBinTree)


//Exercise 18: Write a function foldInOrder : ((a, b) -> b) -> b ->
    a
//BinTree -> b that makes an in-order traversal of the tree and folds over the
    elements.
let foldInOrder f acc tree =
    List.fold f acc (inOrder tree)

printfn "%A" (foldInOrder (fun n m-> printfn "%A + %A = %A" n m (n+m); n + m) 0
    intBinTree)

//Exercise 19: Extend the lecture s expression language with the additional
    operators: max, min, and
//equals (=). They all take two arguments. The equals operator should return 1
    when true, and 0 when
//false.
type expr =
    | Const of int
    | If of expr * expr * expr
    | Prim of string * expr * expr
    | Var of string
    | Bind of string * expr * expr

let rec replace scope str value =
    match scope with
    | Var(x) when x = str -> value
    | Var(x) -> Var(x)
    | Prim(opr, expr1, expr2) -> Prim(opr, (replace expr1 str value), (replace
      expr2 str value))
    | If(cond, expr1, expr2) -> If((replace cond str value), (replace expr1 str
      value), (replace expr2 str value))
    | Bind(str', expr1, expr2) when str<>str' -> Bind(str', (replace expr1 str
      value), (replace expr2 str value))
    | Bind(str', expr1, expr2) -> Bind(str', expr1, expr2)
    | Const i -> Const i

let rec eval expr =
    match expr with
    | Const i -> i
```

```
    | Prim("-", expr1, expr2) -> eval expr1 - eval expr2
    | Prim("+", expr1, expr2) -> eval expr1 + eval expr2
    | Prim("max", expr1, expr2) -> if(eval expr1 > eval expr2) then eval expr1
      else eval expr2
    | Prim("min", expr1, expr2) -> if(eval expr1 > eval expr2) then eval expr2
      else eval expr1
    | Prim("=", expr1, expr2) -> if((eval expr1) = (eval expr2)) then 1 else 0
    | Prim(opr, _, _) -> (printfn "Operation %s not supported" opr; 0)
    | If(cond, _, expr) when eval cond = 0 -> eval expr
    | If(expr1, expr2, _) -> eval expr2
    | Bind(str, value, scope) -> eval <| replace scope str value
    | _ -> 0


//Exercise 20: Write more example expressions in the lecture s expression
    language (in abstract
//syntax), and evaluate them using the eval function.
printfn "%A" (eval (Const 5))
printfn "%A" (eval (Prim ("+", Const 12, Const 5)))
printfn "%A" (eval (Prim("+", Prim("max", Const 20, Const 10), (Prim("=", Const
    -12, (Prim ("min", Const -12, Const -5)))))))

//Exercise 21: Extend the expression language with conditional expressions If(e1,
    e2, e3)
//corresponding to C#  s expression e1 ? e2 : e3 or F#  s expression if e1 then
    e2 else e3.
//The type (abstract syntax) for expressions may be extended as follows:
//type expr =
//Const of int
//| If of expr * expr * expr
//| Prim of string * expr * expr
//Extend the interpreter function eval correspondingly. It should evaluate e1, and
     if e1 is non-zero,
//then evaluate e2, otherwise evaluate e3.
printfn "%A" (eval (If(Const(5), Const(3), Const(4))))


//Exercise 22: Extend the expression language with a Bind expression Bind(x, e1,
    e2) where x is
//a variable, e1 is the value bound the x and e2 is the body in which x can be
    referenced. A variable is
//referenced using the constructor Var x.
//The type (abstract syntax) for expressions may be extended as follows:
//type expr2 =
//Const of int
//| If of expr2 * expr2 * expr2
//| Bind of string * expr2 * expr2
//| Var of string
//| Prim of string * expr2 * expr2
//Extend the interpreter function eval correspondingly. It should evaluate e1, and
     bind the result
//value to x and evaluate e2 with new environment.
printfn "%A" (eval (Bind("x", Const(1337), Prim("+", Var "x", Const(3)))))
printfn "%A" (eval (Bind("x", Const(1337), Bind("y", Const(22), Prim("+", Var "x",
     Var "y")))))
printfn "%A" (eval (Bind("x", Const(1337), Bind("x", Const(5), Prim("+", Var "x",
    Var "x")))))
printfn "%A" (eval (Bind("x", Const(1337), Bind("x", Const(5), Prim("+", Var "z",
    Var "q")))))

System.Console.ReadLine() |> ignore
```

D.5 Anders

You can download the source code online here `http://d.pr/U2wu`.

```
F# Exercises
Anders Bech Mellson, anbh@itu.dk

1)
let sqr x = x * x;;

2)
a
let pow (x,n) = System.Math.Pow(x,n);;
let pow x n = System.Math.Pow(x,n);;

b
let pow2 (x,n) = int(System.Math.Pow((float)x,(float)n));;

c
// Virker kun paa positive vaerdier
let rec pow3 x n = if n=0 then 1 else x * pow3 x (n-1);;

// Virker paa negative vaerdier, men den afrunder pga int
let rec pow3 x n = if n=0 then 1 else if n>0 then x * pow3 x (n-1) else pow3 x (n
    +1)/x;;

// Med float, virker nu paa negative vaerdier. Kan dog kun kaldes med hele talvae
    rdier som b(eg 1.0, 4.0)
let rec pow3 x n = if n=0.0 then 1.0 else if n>0.0 then x * pow3 x (n-1.0) else
    pow3 x (n+1.0)/x;;

3)
Hvis man bruger +, defaulter den til int. Bruger man ^ defaulter den til string.
let dup x y = x ^ y;;

4)
let rec dupn s n = if n=1 then s else s^dupn s (n-1);;

5)
metode 1
let hourFix(hour) = if hour = 0 then 24 else hour;;
let abs(x:int) = System.Math.Abs(x);;
let max(x:int,y:int) = System.Math.Max(hourFix(x),hourFix(y));;
let min(x:int,y:int) = System.Math.Min(hourFix(x),hourFix(y));;
let timeDiff(h1,m1)(h2,m2) =
        if abs(hourFix(h1)-hourFix(h2))>12 then abs(((max(h1,h2)-24)*60+m1)-(min(
            h1,h2)*60+m2))
        else abs((max(h1,h2)*60+m1)-(min(h1,h2)*60+m2));;

metode 2
let timeDiff (h1, m1)(h2, m2) =
    let abs(x:int) = System.Math.Abs(x) in
    let max(x:int,y:int) = System.Math.Max(x,y) in
    let min(x:int,y:int) = System.Math.Min(x,y) in
    let h1 = if h1 = 0 then 24 else h1 in
    let h2 = if h2 = 0 then 24 else h2 in
    let time1 = ((h1 * 60) + m1) in
    let time2 = ((h2 * 60) + m2) in
    let minutes = if abs(h1-h2)>12 then abs((max(time1,time2)-1440) - min(time1,
        time2)) else abs(max(time1,time2) - min(time1,time2)) in
    (minutes)
in;;
```

```
6)
let minutes(h1,m1) = timediff(24,00)(h1,m1);;

7)
let rec downTo(x:int) = if x = 1 then 1::[] else x::downTo(x-1);;

let rec downTo2(x:int) = match x with 1 -> 1::[] | x -> x::downTo2(x-1);;

8)
let removeEven xs:list<int> = [for i in 1 .. List.length xs do if i % 2 = 1 then
    yield xs.[i-1]];;

9)
metode 1
let combinePairs (xs:list<int>) = [for i in 0 .. (List.length xs)-2 do if i%2=0
    then yield (xs.[i],xs.[i+1])];;

metode 2
let rec combinePairs xs = match xs with [_] -> [] | x::y::xr -> (x, y) ::
    combinePairs(xr);;

10)
let explode (s:string) = List.ofArray (s.ToCharArray());;

let rec explode2 (s:string) = if s.Length=0 then [] else s.[0] :: explode2(s.
    Remove(0,1));;

11)
a
let rec foldr f b xs = match xs with [] -> b | x::xs -> f(x, foldr f b xs);;

metode 1
let implode (c:list<char>) = foldr (fun (char,x) -> char.ToString() + x) "" c;;

metode 2
let implode (c:list<char>) = List.fold (fun x char -> x + char.ToString()) "" c;;

metode 3
let implode (chars:list<char>) = new string [|for c in chars -> c|];;

b
let implode2 (c:list<char>) = List.foldBack (fun acc elem -> acc.ToString()+elem)
    c "";;

c
let rec foldl f b xs = match xs with [] -> b | x::xs -> foldl f (f(x,b)) xs;;
let implodeRev (c:list<char>) = foldl (fun (char,x) -> char.ToString() + x) "" c;;

d
let implodeRev2 (c:list<char>) = List.fold (fun x char -> char.ToString() + x) ""
    c;;

12)
a
let toUpper (s:string) = s.ToUpper();;

b
let stringToChar (s:string) = s.ToCharArray();;
let charArrayToUpperString (chars:char[]) = new string [|for c in chars -> System.
    Char.ToUpper(c)|];;
let toUpper1  = stringToChar >> charArrayToUpperString;;
```

```
c
let stringToChar (s:string) = s.ToCharArray();;
let charArrayToUpperString (chars:char[]) = new string [|for c in chars -> System.
    Char.ToUpper(c)|];;
let toUpper2  = charArrayToUpperString << stringToChar;;

d
let toUpper3 (s:string) = s |> stringToChar |> charArrayToUpperString;;

13)
let palindrome (s:string) = (new string[|for i in s.Length-1..-1..0 -> s.[i]|]).
    ToUpper() |> (fun x -> x = s.ToUpper());;

14)
let aboveZero x = x>0;;
let rec ack (m,n) = match (m,n) with
                                | (0,_) -> n+1
                                | (aboveZero,0) -> ack(m-1,1)
                                | aboveZero -> ack(m-1,ack(m,n-1));;
Result:
> ack (3,11);;
val it : int = 16381

15)
let timeArg1 f a =
    let start = System.DateTime.Now in
    let res = f (a) in
    let finish = System.DateTime.Now in (res, finish - start);;

16)
type 'a BinTree =
    Leaf
  | Node of 'a * 'a BinTree * 'a BinTree;;

let rec inOrder tree =
  match tree with
    Leaf -> []
    | Node(n,treeL,treeR) ->
    inOrder treeL @ n :: inOrder treeR;;

17)
metode 1
let rec mapInOrder f tree =
  match tree with
    | Leaf -> Leaf
    | Node(n,treeL,treeR) ->
      let left = mapInOrder f treeL
      let nn = f n
      let right = mapInOrder f treeR
      Node(nn, left, right);;

metode 2 der returnerer en liste
let half x = x/2;;
let mapInOrder tree = List.map half (inOrder tree);;

Depending on the type of function you use in the map you could get different
    result from running through the tree in different directions. Eg if you bring
    a value with you it will have different impacts depending on the direction.

18)
let rec foldl f b xs = match xs with [] -> b | x::xs -> foldl f (f(x,b)) xs;;
```

```
let foldInOrder f acc tree = tree |> inOrder |> foldl f acc;;

19)
type expr =
  Const of int
  | Prim of string * expr * expr;;

let rec eval expr =
  match expr with
    Const i -> i
  | Prim("-", expr1, expr2) -> eval expr1 - eval expr2
  | Prim("+", expr1, expr2) -> eval expr1 + eval expr2
  | Prim("=", expr1, expr2) -> if eval expr1 = eval expr2 then 1 else 0
  | Prim("min", expr1, expr2) -> System.Math.Min(eval expr1, eval expr2)
  | Prim("max", expr1, expr2) -> System.Math.Max(eval expr1, eval expr2)
  | Prim(opr, _, _) -> (printfn "Operation %s not supported" opr; 0);;

20)
eval (Prim ("=", Const 32, Const 51));;
eval (Prim ("=", Const 51, Const 51));;
eval (Const(34));;

21)
type expr =
    | Const of int
    | If of expr * expr * expr
    | Prim of string * expr * expr;;

let rec eval expr =
  match expr with
    Const i -> i
  | Prim("-", expr1, expr2) -> eval expr1 - eval expr2
  | Prim("+", expr1, expr2) -> eval expr1 + eval expr2
  | Prim("=", expr1, expr2) -> if eval expr1 = eval expr2 then 1 else 0
  | Prim("min", expr1, expr2) -> System.Math.Min(eval expr1, eval expr2)
  | Prim("max", expr1, expr2) -> System.Math.Max(eval expr1, eval expr2)
  | Prim(opr, _, _) -> (printfn "Operation %s not supported" opr; 0)
  | If(expr1,expr2,expr3) -> if eval expr1 <> 0 then eval expr2 else eval expr3;;

22)
type expr =
  Const of int
  | If of expr * expr * expr
  | Bind of string * expr * expr
  | Var of string
  | Prim of string * expr * expr;;

let rec lookup env y =
  match env with
  | [] -> failwith ("Unknown variable: " + y)
  | (x,v) :: rest -> if x=y then v else lookup rest y;;

let rec eval expr env =
  match expr with
    Const i -> i
  | Prim("-", expr1, expr2) -> eval expr1 env - eval expr2 env
  | Prim("+", expr1, expr2) -> eval expr1 env + eval expr2 env
  | Prim("=", expr1, expr2) -> if eval expr1 env = eval expr2 env then 1 else 0
  | Prim("min", expr1, expr2) -> System.Math.Min(eval expr1 env, eval expr2 env)
  | Prim("max", expr1, expr2) -> System.Math.Max(eval expr1 env, eval expr2 env)
  | Prim(opr, _, _) -> (printfn "Operation %s not supported" opr; 0)
```

```
| If(expr1,expr2,expr3) -> if eval expr1 env <> 0 then eval expr2 env else eval
    expr3 env
| Bind(string,expr1,expr2) -> eval expr2 <| (string,eval expr1 env)::env
| Var(string) -> lookup env string;;
```

Appendix E

Responsible persons

This is a list of our individually responsible areas of the project. We all have many overlaps, so this is not a completely accurate picture - only a birds view on which key areas we each have been responsible for.

- **Ahmad**
  - Test manager.
  - Coding the plurality of web services.
  - Coding following things in the client
    - Forms-to-template conversion
    - In collaboration with Martin:
      - Authentication and account management
      - Charting and statistics of surveys
  - Sections of this report
    - Technical description – Client
    - Testing
    - Code improvement
- **Martin**
  - Coding of some web services and tests
  - Coding following things in the client
    - Help section
    - In collaboration with Ahmad:
      - Authentication and account management
      - Charting and statistics of surveys
  - Sections of this report
    - Requirements
    - User manual
- **Nicolai**

  - Contact person between the group and the course-manager
  - Sections of this report
    * Problem explanation and background
    * Interface design analysis
    * Integration tests
  - Coding the following things in the client
    * Layout
    * Survey listing
    * Survey creation-page
    * Take Survey
    * Integration test bug-fixes

- **Mark**
  - Server master/admin.
  - Coding of some web services.
  - Coding following things in the client
    * Participant management module
  - Testing of parts of Server and Client.
  - Sections of this report
    * Problem Analysis

- **Anders**
  - Project Manager.
  - Communication and agreements with SMU.
  - Example code of all web services.[1]
  - Coding of some web services.
  - Setup of reports in LaTeX.
  - Sections of this report
    * Preface and introduction
    * Data Model
    * Documentation of communication
    * Appendixes, Front page, Layout etc

[1] See the sample code online here.`http://d.pr/xhGf`

Appendix F

Group constitution

Unfortunately our constitution in danish. But we have added it anyway, hoping that the danish language is ok in this part. The constitution follows on the next pages.

RSVP

Vi skal lave

- Server
    - Skal understøtte et domæne (f. eks. test, statistik, afstemninger, planlægning)
- Klient
    - Skal kunne kommunikere med serveren, serveren skal ikke miste funktionalitet som Singaporianerne benytter
- Beskrivelse af serveren og hvordan den skal bruges
    - SOAP? REST?
- Datamodel
    - **Deadline:** 1. Marts inkl. testdata
- Rapport - vigtig

Gruppekonstitution

Organisation

Roller (ansvarlige)	Klient - Mark/Martin Server - Nicolai/Ahmad Testing - Ahmad Dokumentation (UML/kodekommentarer) - Martin Teknik (server-admin) - Mark Projektleder - Anders Kontaktperson - Nicolai
Rotation	Foretages løbende efter behov
Mødested	IT-Universitet 3. sal
Diskussioner	Tidsbegrænsning med skiftende ordstyrer
Dokumentation	XMLDoc til metodesignaturer. UML til overordnet programstruktur (såsom F# moduler og namespaces). Når vi kender F# bedre kan vi bestemme hvordan vi dokumenterer systemet overordnet.

	Arbejdsblade til større avancerede komponenter Referat fra møder Aftaler (fx efter diskussion) Korrekt kildehenvisning

Arbejde

Ambitions niveau	10+
Arbejdstider	Forelæsning: 3 - 4 timer efter slut Normal: 10:00-16:00
Beslutninger	Flertalsbestemning med tidsgrænse

Samvær

Sanktioner	En 50'er bliver tilføjet til en pulje hvis man kommer mere for sent en 15 minutter uden at have givet lyd fra sig. Puljen kan eventuelt blive brugt til at spise sammen efter projektet er færdigt.

Kode konventioner

- Projektledelsesmodel / udviklingsmodel
    - Hvilken model arbejder vi ud fra
    - Fase opdelt?

Design

- Brugere
- Unikke link som passer til en email

Roller

- Mark Philip Andersen
    - mpan@itu.dk / osiris@mail.dk

    - 2984 7167
- Martin Jeanty Larsen
    - 3074 7977
    - martin@jeanty.dk
- Anders Bech Mellson
    - 2211 6649
    - anders@bechmellson.com
- Nicolai Bo Skovvart
    - 2848 6086
    - skovvart@gmail.com
- Ahmad Salim Al-Sibahi
    - 2521 3121
    - asal [at] itu [dot] dk, asal.thegeek [at] gmail [dot] com

Testing

Vi tester hver gang en ny funktion er tilføjet.
Skal med fra starten.

Milestones

- Datamodel
    - kig på Tofte's for inspiration
- Opsætning af versionskontrol
    - Team Foundation Server
        - Codeplex
        - Vores egen

Individuelle opgaver

Start on your individual profile on the Wiki
Read about the RSVP Case at ITU.

Appendix G

Web services

On the following pages you can find a list over the web services we agreed with SMU to implement on our server. This version of the list is from the 22.03.2011.

Struct	Struct description
Invitee(string email,string name)	A struct describing an invitee.
Host(string email, string password, string name)	A struct describing a host. Can contain partial null values.
Survey(string title, int ID)	A struct describing a survey, name with ID.

Webservice	Webservice description
bool RSVPService.CreateHost(Host host)	Adds a new host to the webservice. host requires email, password. Optional parameters: name. Throws Exception(string s) when unable to create host.
bool RSVPService.DeleteHost(Host host)	Deletes a host from the database. host requires email, password. Optional parameters: name. Throws Exception(string s) when unable to delete host.
bool RSVPService.AuthenticateHost(Host host)	Authenticates a host - checks if the email/password combination corresponds to what is stored in the database.
int RSVPService.CreateSurvey(string template, Host host)	Create a survey based on template. host requires email, password. Returns surveyId. Throws Exception(string s) when a Host with a blank or invalid

	password tries to create a survey. Also throws an exception if the TEMPLATE cannot be parsed correctly.
bool RSVPService.DeleteSurvey(int surveyId, Host host)	Delete survey with id surveyId. host requires email, password. Throws Exception(string s) if it's not the surveys owner who tries to delete the survey.
bool RSVPService.AddInviteeToSurvey (int surveyId, string inviteeMail, Host host)	Add an invitee as a participant to survey with id surveyId. participant requires email, host requires email, password. Throws Exception(string s) if unable to authenticate host.
bool RSVPService.RemoveInviteeFromSurvey (int surveyId, string inviteeMail, Host host)	Remove a participant from a survey. participant requires email, host requires email, password. Throws Exception(string s) if unable to authenticate host.
Invitee[] RSVPService.ViewInviteesFromSurvey (int surveyId, Host host)	Get an array of Invitee's for the given survey. Throws Exception(string s) if unable to authenticate host.
string RSVPService.GetSurveyQuestions (int surveyId)	Show generated XML for survey with id surveyId.
string RSVPService.GetSurveyResults (int surveyId, Host host)	Show statistics for survey with id surveyId. The host requires password and email. Throws Exception(string s) if unable to authenticate host.
Survey[] RSVPService.ViewSurveys (Host host)	Lists the surveys a host has access to. Throws Exception(string s) if unable to authenticate host.
bool RSVPService.SubmitQuestionAnswer (int optionID, string answer, string inviteeEmail)	Answer question with the option that has id optionID. If a text-question, answer will contain the answer, if not answer can be

	an empty string or null. invitee requires email. Throws Exception(string s) if invitee does not have access to the survey containing the question.
bool RSVPService.RemoveQuestionAnswer (int optionID, string inviteeEmail)	Answer question with the option that has id optionID. If a text-question, answer will contain the answer, if not answer can be an empty string or null. invitee requires email. Throws Exception(string s) if invitee does not have access to the survey containing the question.
bool RSVPService.SubmitSurveyAnswers (int SurveyID, string inviteeEmail)	Call this when an invitee is finished with the survey. Will return exceptions if mandatory questions are not answered (describing which questions need answering).
bool RSVPService.DeleteGuestlist(string name, Host host)	Deletes a guest list and all invitees associated. Returns true if successful and an exception if not.
bool RSVPService.CreateGuestlist(string name, Host host)	Creates a new guest list associated with host. host requires email, password. Returns true if successful and an exception if not.
bool RSVPService.AddInviteeToGuestList(string guestListName, Host host, string inviteeEmail, string inviteeName)	Adds an invitee to host's guestlist with name guestListName. invitee requires email, host requires email, password. Throws Exception(string s) if unable to authenticate host, or if guestListName doesn't exist.
bool RSVPService.RemoveInviteeFromGuestList(string guestListName, Host host, string inviteeEmail)	Removes a invitee from host's guestlist with name guestListName. invitee requires email, host requires email, password. Returns true if successful and an exception if not.

string[] RSVPService.ViewGuestLists(Host host)	Shows the name of all guestlists associated with host. Host requires email, password. Optional parameters: Host.name Returns string[] if successful and an exception if not. return exception if Host has no guest list (exception message: “No guest lists available”)
Invitee[] RSVPService.ViewInviteesFromGuestList (string guestListName, Host host)	Show all invitees belonging to a host’s guestlist with name name. Host requires email, password. Returns invitee[] if successful and an exception if not. return exception if list has no invitees (exception message: “No invitees available”)
bool RSVPService.ChangeHostPassword(Host host, string newPassword)	Change the password for host. Host requires email, password. Password can be null or an emptystring, but newPassword must not be null or an emptystring. Throws Exception(string s) if unable to validate host.

Appendix H

Testing plans

H.1 Server unit tests

Test plan - Server

GuestListTest

Shared Data
Service : A default RSVPService object
Host: A Host object created with following email=”Aa@bb.cc” and password=”12512”
SurveyId : The id of the survey created with following template: “template main = begin end”

MethodName: bool CreateGuestList(string name, Host host)

Input	Expected output	Actual output	Explanation
name = “testguestlist” host = Host{Legit@host.dk, 1234}	True	True	Create a guest list on a valid host.
name = “testguestlist” host = Host{“NonLegit@host.dk”,“1234”}	RSVPException	RSVPException	Create a guest list on a non existing host.
name = “testguestlist” host = Host{“Legit@host.dk”,“4321”}	RSVPException	RSVPException	Create guest list on a valid host with wrong password.

MethodName: bool DeleteGuestList(string name, Host host)

Input	Expected output	Actual output	Explanation
name = “testguestlist” host = Host{Legit@host.dk,	True	True	Delete existing guest list.

name = “testguestlist” host = Host{“NonLegit@host.dk”,“1234”}	RSVPException	RSVPException	Delete guestlist on non existing host.
name = “NonExistingGuestList” host = Host{“Legit@host.dk”,“1234”}	RSVPException	RSVPException	Delete non existing guestlist on a valid host.

MethodName: bool ViewGuestLists(Host host)

Input	Expected output	Actual output	Explanation
host = Host{Legit@host.dk, 1234}	True	True	View guest lists
host = Host{“NonLegit@host.dk”,“1234”}	True	True	View non existing guestlist
host = Host{“NonLegit@host.dk”,“1234”}	RSVPException	RSVPException	View guestlist on non existing host

InviteeGuestListTest

Shared Data

Service : A default RSVPService object
Host: A Host object created with following email=”Aa@bb.cc” and password=”12512”
SurveyId : The id of the survey created with following template: “template main = begin end”

MethodName: bool AddInviteeToGuestList(string guestListName, Host host, string inviteeEmail, string inviteeName)

Input	Expected output	Actual output	Explanation

guestListName = “testguestlist” host = Host{Legit@host.dk, 1234} inviteeEmail = “herp@derp.com” inviteeName= ”herp derp”	True	True	Add invitee to guestlist.
guestListName = “fakeguestlist” host = Host{Legit@host.dk, 1234} inviteeEmail = “herp@derp.com” inviteeName= ”herp derp”	RSVPException	RSVPException	Add invitee to non existing guestlist.
guestListName = “fakeguestlist” host = Host{NonLegit@host.dk, 1234} inviteeEmail = “herp@derp.com” inviteeName= ”herp derp”	RSVPException	RSVPException	Add invitee to existing guestlist with a non existing host.
guestListName = “testguestlist” host = Host{Legit@host.dk, 4321} inviteeEmail = “herp@derp.com” inviteeName= ”herp derp”	RSVPException	RSVPException	Add invitee to existing guestlist with an existing host with wrong password.

MethodName: bool ViewInviteesFromGuestList(string name, Host host)

Input	Expected output	Actual output	Explanation

name = “testguestlist” host = Host{Legit@host.dk, 1234}	True	True	View invitees from a guestlist.
name = “nonlegitguestlist” host = Host{“NonLegit@host.dk”,“1234”}	RSVPException	RSVPException	View invitees from non existing guest list.

MethodName: bool RemoveInviteeFromGuestList(string guestListName, Host host, string inviteeEmail)

Input	Expected output	Actual output	Explanation
guestListName = “testguestlist” host = Host{“Legit@host.dk”, “1234”} inviteeEmail =”herp@derp.com”	True	True	Remove an invitee from a guestlist.
name = “fakeguestlist” host = Host{“NonLegit@host.dk”,“1234”} inviteeEmail =”herp@derp.com”	RSVPException	RSVPException	Remove an invitee from a non existing guestlist.
guestListName = “testguestlist” host = Host{“Legit@host.dk”, “1234”} inviteeEmail =”herp@derp.com”	RSVPException	RSVPException	Remove non existing invitee from a guestlist.
guestListName = “testguestlist” host = Host{“NonLegit@host.dk”, “1234”} inviteeEmail =”herp@derp.com”	RSVPException	RSVPException	Remove an invitee from a guestlist on a non existing host.

HostTest

Shared Data

Service : A default RSVPService object
validHost: A Host object created with following email=”valid-google.email@gmail.com” and password=”1234”
hostWrongPassword: A Host object created with following email=”valid-google.email@gmail.com” and password=”wrongPassWord”
validHost is created on the webservice.

Method: bool AuthenticateHost(Host host)

Input	Expected output	Actual output	Explanation
Host { email = “test@email.com”, password = “1234” }	false, true	false, true	Authenticate host before it is created on the webservice -> Authenticate after host is created on the service
hostWrongPassword	false	false	Wrong password
Host { Email = "does@not.exist", Password = "speedy123" }	false	false	Host doesn’t exists
Host { Email = "does@not.exist" }	false	false	Authentication with password not specified
Host { Email = “” }	false	false	Wrong email-format, no password
Host { Email = null }	false	false	Wrong email (null), no password
Host { }	false	false	No email or password specified

Method: bool CreateHost(Host host)

Input	Expected output	Actual output	Explanation

Host { Email = "test@email.com", Password = "1234" }	true	true	Create a host that doesn't exist in the database already
Host { Email = "bad@example.email", Password = "*****" }	RSVPException	RSVPException	Invalid email, host-top-level-domain is too long to be considered valid
Host { Email = "bad@example.e", Password = "*****" }	RSVPException	RSVPException	Invalid email, host-top-level-domain is too short to be considered valid
Host { Email = ".bad@example.com", Password = "*****" }	RSVPException	RSVPException	Invalid email, local name must not start or end with a dot.
Host { Email = "bad.@example.com", Password = "*****" }	RSVPException	RSVPException	Invalid email, local name must not start or end with a dot.
Host { Email = "ba..d@example.com", Password = "*****" }	RSVPException	RSVPException	Invalid email, local name must not have two subsequent dots.
Host { Email = "test@exa..mple.com", Password = "*****" }	RSVPException	RSVPException	Invalid email, hostname must not have two subsequent dots.
Host { Email = "test@.exa.mple.com", Password = "*****" }	RSVPException	RSVPException	Invalid email, hostname must not start with a dot.
Host { Email = "test@exa.mple..com", Password = "*****" }	RSVPException	RSVPException	Invalid email, hostname must not end with a dot.
validHost	RSVPException	RSVPException	Creating a host that already exists.
Host { email="valid-google.email@gmail.com" and password="different password" }	RSVPException	RSVPException	Creating a host with an email that is already taken by another host
Host { Email = "valid@ema.il", Password = "" }	RSVPException	RSVPException	Creating a host with an invalid password
Host { Email = "valid@ema.il", Password = null }	RSVPException	RSVPException	Creating a host with an invalid password

Host { Email = "valid@ema.il" }	RSVPException	RSVPException	Creating a host with an unspecified password

Method: bool DeleteHost(Host host)

Input	Expected output	Actual output	Explanation
validHost	true	true	Deleting a host that exists with the proper email/ password combination
hostWrongPassword	RSVPException	RSVPException	Deleting a host that exists with a bad email/password combination
Host { Email = "does@notex.ist", Password = "wrong password" }	RSVPException	RSVPException	Deleting a host that doesn't exist

Method: bool ChangeHostPassword(Host host, string newPassword)

Input	Expected output	Actual output	Explanation
validHost, "new password"	true	true	Changing a hosts password with a proper old email/ password combination and a new password
Host { Email = "does@notex.ist", Password = "old password" }, "new password"	RSVPException	RSVPException	Host doesn't exist
hostWrongPassword, "new password"	RSVPException	RSVPException	Old password didn't match up for host
validHost, ""	RSVPException	RSVPException	New password is invalid
validHost, null	RSVPException	RSVPException	New password is invalid

SurveyTest

Shared Data

Service : A default RSVPService object
Host: A Host object created with following email=”martin@jeanty.dk” and password=”qwerty42millarder”
SurveyId : The id of the survey created with following template: “template main = begin end”

Method Name: bool CreateSurvey(string template, Host host)
Method General Data:

Survey with no questions: template emtpy_test_survey = begin end
Survey with mixed questions: template mandatory_mix_test_survey = begin
mandatory select "Do you like testing?" ["Yes","No"]
explanation "You may not like testing, but"
checkboxes "What type of testing do you prefer?"
["Whitebox","Blackbox", "Unit", "Automated", "Contracted"]
text "How much do you like testing?"
end

Input	Expected output	Actual output	Explanation
template: survey with no questions Host: default	true	true	A valid template with no questions should be regarded as valid
template: survey with mixed questions Host: default (created once)	true	true	A survey with valid template should not give an error
template: survey with mixed questions Host:default (created twice)	true	true	There should be no restrictions on creating two surveys with the same questions
template: none given Host:default	RSVPException	RSVPException	One cannot create a survey without a template description
template: “” Host: default	RSVPException	RSVPException	One cannot create a survey without a template description
template: invalid Host: default	RSVPException	RSVPException	One cannot create a survey with invalid template

template: survey with mixed questions Host: invalid	RSVPException	RSVPException	One cannot create a survey with an invalid host

Method Name: string GetSurveyQuestions(int surveyId)
Method General Data:

Survey with no questions: template emtpy_test_survey = begin end

Survey with mixed questions:
```
template mandatory_mix_test_survey = begin
                                        mandatory select "Do you like testing?" ["Yes","No"]
                                    explanation "You may not like testing, but"
                                     checkboxes "What type of testing do you prefer?"
                                           ["Whitebox","Blackbox", "Unit",  "Automated", "Contracted"]
                                    text "How much do you like testing?"
                                    end
```

Input	Expected output	Actual output	Explanation
survey: survey with no questions	valid survey-question XML	valid survey-question XML	
survey: survey with mixed questions	valid survey-question XML	valid survey-question XML	
suvey: non-existing survey	RSVPException	RSVPException	

Method Name: string GetSurveyResults(int surveyId, Host host)
Method General Data:

Survey with mixed questions:
```
template mandatory_mix_test_survey = begin
                                        mandatory select "Do you like testing?" ["Yes","No"]
                                    explanation "You may not like testing, but"
                                     checkboxes "What type of testing do you prefer?"
                                           ["Whitebox","Blackbox", "Unit",  "Automated", "Contracted"]
                                    text "How much do you like testing?"
                                    end
```

Input	Expected output	Actual output	Explanation
survey: survey with mixed questions host: default	valid XML	valid XML	
survey: non-existing survey host: default	RSVPException	RSVPException	

survey: survey with mixed questions host: invalid	RSVPException	valid XML (**FIXED**)	**Error**: the email was checked with itself instead of checked with database when checking ownership

Method Name: bool DeleteSurvey(int surveyId, Host host)

Input	Expected output	Actual output	Explanation
surveyId = default email = “martin@jeanty.dk” password = “qwerty42millarder”	true	true	
surveyId = 666 email = “martin@jeanty.dk” password = “qwerty42millarder”	RSVPException	RSVPException	
surveyId = default email = “evilguy@hell.org” password = “666”	RSVPRSVPExcepti on	true (**FIXED**)	Error in SQL checking that the given host has admin role to the specified survey.

MethodName: Survey[] ViewSurveys(Host host)

Input	Expected output	Actual output	Explanation
Shared host	true	true	5 surveys created by host - cheks that all created surveyId’s from the created surveys are contained in the returned array.
Shared host	RSVPException	RSVPException	5 surveys created by host then deleted again before method is called
Shared host	RSVPException	RSVPException	No surveys created by host

SurveyInviteeTest

Shared Data

Service : A default RSVPService object
validHost = Host { Email = "host@example.com", Password = "batman123", Name = "Senõr Host" }
wrongHost = Host { Email = "another@email.com", Password = "doctor x", Name = "Matt Damon" }
invalidHost = Host { Email = "host@example.com", Password = "supergirl", Name = "Senõr Host" }
inviteeEmail = “dan@the.man”

surveyId = the id of a created survey
validHost and wrongHost are created on the webservice.

MethodName: bool AddInviteeToSurvey(int surveyId, string inviteeEmail, Host host)

Input	Expected output	Actual output	Explanation
surveyId, inviteeEmail, validHost	true	true	Add a valid email to a valid survey
surveyId, inviteeEmail, wrongHost	RSVPException	RSVPException	Valid host, but not host for the survey, tries to add a participant to the survey.
surveyId, inviteeEmail, invalidHost	RSVPException	RSVPException	Host doesn't exist, and isn't the host of the survey, tries to add a participant to the survey
surveyId, "dan@the.animal", validHost	RSVPException	RSVPException	Bad email-syntax for inviteeEmail
surveyId, inviteeEmail, validHost surveyId, inviteeEmail, validHost	RSVPException	RSVPException	Invitee already added to survey

MethodName: bool RemoveInviteeFromSurvey(int surveyId, string inviteeEmail, Host host)

Input	Expected output	Actual output	Explanation
surveyId, inviteeEmail, validHost	true	true	Remove an existing invitee from a survey that has said invitee, where host is the host of they survey.
surveyId, inviteeEmail, validHost	RSVPException	RSVPException	Remove an invitee that does not exist on the survey.
surveyId, inviteeEmail, wrongHost	RSVPException	RSVPException	Host that isn't host of survey tries to remove an invitee from survey.

MethodName: Invitee[] ViewInviteesFromSurvey(int surveyId, Host host)

Input	Expected output	Actual output	Explanation
surveyId, validHost	{ Invitee{ Email = "host@example.com" }, Invitee{ Email = "dan@the.man" } }	{ Invitee{ Email = "host@example.com" }, Invitee{ Email = "dan@the.man" } }	Output from an existing survey where host is the host of the survey.
surveyId, wrongHost	RSVPException	RSVPException	Wrong host tries to view output

surveyId, invalidHost	RSVPException	RSVPException	Non-existing (and wrong) host tries to view output
-1, validHost	RSVPException	RSVPException	survey doesn't exist.

SurveyAnswerTest

Shared Data

Service : a default RSVPService object to call our web service methods
Host : a Host object with email: "Hans_pilgård@gmail.com" and password: "Lise"
Invitee : an Invitee object with email: "Bo_Michaelsen@live.dk" and name: "Bo Michaelsen"
Empty Survey: A survey with no questions
Ordinary Survey: A survey with no mandatory questions and mixed questiontypes(select,checkboxes and text)
Mandatory Mix Survey: A survey with some mandatory questions and mixed questiontypes(select,checkboxes and text)
Mandatory Only Survey: A survey with only mandatory questions and mixed questiontypes(select,checkboxes and text)
No Access Survey: Like ordinary survey but the invitee has no access to it

Method Name: bool SubmitSurveyAnswers(int surveyId, string inviteeEmail)
General Comment: This method it tested along with help of SubmitQuestionAnswer and RemoveQuestionAnswer as an individual test won't make sense

Input	Expected output	Actual output	Explanation
Empty survey Invitee email	true	true	Try to submit an empty survey, as it has no questions (and thus no mandatory questions unanswered) submission should be fine
Ordinary survey Invitee email (No answers submitted)	true	true	As there are no mandatory questions in ordinary survey, one should be able to submit it without any answer
Ordinary survey Invitee email (Some answers submitted)	true	true	As there are no mandatory questions in ordinary survey, one should be able to submit it without answering all questions
Ordinary survey Invitee email (Some answers submitted and some answers deleted)	true	true	Removing nonmandatory answers should not make a difference when submitting a survey
Ordinary survey Invitee email (All answers submitted)	true	true	A completed survey should allow a submission

Ordinary survey Invitee email (All answers submitted and some answers deleted)	true	true	Removing nonmandatory question's answers should not make a difference when submitting a survey, even if all answers were submitted.
Ordinary survey Invitee email (All answers submitted and then deleted again)	true	true	Same as above
Ordinary survey Invitee email (Two submissions)	true	true	For better reliability survey submission should be idempodent
No Access survey Invitee email	RSVPException	RSVPException	One should not be able to submit a survey one has not access to
Mandatory mix survey Invitee email (No answer submitted)	RSVPException	RSVPException	One should not be able to submit a survey without answering all mandatory questions
Mandatory mix survey Invitee email (No mandatory answer submitted)	RSVPException	RSVPException	One should not be able to submit a survey without answering all mandatory questions even though one has submitted other questions
Mandatory mix survey Invitee email (No ordinary answer submitted)	true	true	One should be able to submit a survey with all mandatory questions answered, even though not all questions has been answered
Mandatory mix survey Invitee email (All anwers submitted)	true	true	A completed survey should allow a submission
Mandatory only survey Invitee email (No answer submitted)	RSVPException	RSVPException	One should not be able to submit a survey without answering all mandatory questions
Mandatory only survey Invitee email (Some answers submitted)	RSVPException	RSVPException	One should not be able to submit a survey without answering all mandatory questions
Mandatory only survey Invitee email (All answers submitted)	true	true	A completed survey should allow a submission
Mandatory only survey Invitee email (All answers submitted and some deleted)	RSVPException	RSVPException	One should be able to submit a survey with all mandatory questions answered, even though they were answered and then deleted

Method Name: bool SubmitQuestionAnswer(int optionId, string answerText, string inviteeEmail)

Input	Expected output	Actual output	Explanation
optionId: Valid option Id inviteeEmail: Invitee email (submitted once)	true	true	A normal submission should work fine
optionId: Valid option Id inviteeEmail: Invitee email (submitted twice)	true	RSVPException (**FIXED**)	Submit question answer should be idempodent and thus work with multiple submissions
optionId: Invalid options Id(-1) inviteeEmail: Invitee email	RSVPException	RSVPException	One should not be able to submit an answer to an question that doest not exist
optionId: Valid option Id answerText: “100” inviteeEmail: Invitee email (answering a text question)	true	true	One should be able to submit a normal answer to a text question
optionId: Valid option Id answerText: **null** inviteeEmail: Invitee email (answering a text question)	RSVPException	RSVPException	One should not be able to submit an answer with no text to a text based question
optionId: Valid option Id answerText: “” inviteeEmail: Invitee email (answering a text question)	RSVPException	RSVPException	One should not be able to submit an answer with no text to a text based question
optionId: Valid option Id answerText: “ \n \t \r\n \r” inviteeEmail: Invitee email (answering a text question)	RSVPException	RSVPException	One should not be able to submit an answer with whitespaces to a text based question

Method Name: bool RemoveQuestionAnswer(int optionId, string inviteeEmail)

Input	Expected output	Actual output	Explanation
optionId: Valid option Id inviteeEmail: Invitee email	true	true	A normal removal should work fine
optionId: Invalid option Id (-1) inviteeEmail: Invitee email	RSVPException	RSVPException	Removing an answer for a question one has not access to should not be

			allowed
optionId: Valid option Id inviteeEmail: Invitee email (question not answered)	true	RSVPException (**FIXED**)	Removing an answer for a question one has not answered should be ok, and idempotent if one has already removed an answer to a question one has answered
optionId: Valid option Id inviteeEmail: Invitee email (survey has been submitted)	RSVPException	RSVPException	One cannot modify a questions answer after one has submitted the survey

H.2 Client unit tests

Models

ModelName: AuthenticationModels_ChangePasswordModel

Input	Expected output	Actual output	Explanation
ChangepasswordModel OldPassword = “test” NewPassword = “test2” ConfirmNewPassword = “test2”	True	True	Positive test
ChangepasswordModel OldPassword = “test” NewPassword = “test2” ConfirmNewPassword = “test3”	True	True	Negative test, NewPassword and ConfirmNewPassword does not match
ChangepasswordModel OldPassword = null NewPassword = “test2” ConfirmNewPassword = “test2”	True	True	Negative test, empty field
ChangepasswordModel OldPassword = “test” NewPassword = “test2” ConfirmNewPassword = null	True	True	Negative test, empty field and non matching NewPassword and ConfirmNewPassword

ModelName: AuthenticationModels_LoginModel

Input	Expected output	Actual output	Explanation
LoginModel Email = “@test.dk” Password = “1234” RememberPassword = true	True	True	Negative test, invalid email format

LoginModel Email = “test@test.dk” Password = null RememberPassword = true	True	True	Negative test, empty field
LoginModel Email = “test@test.dk” Password = “1234” RememberPassword = true	True	True	Positive test

ModelName: AuthenticationModels_CreateModel

Input	Expected output	Actual output	Explanation
CreateModel Email = “test@test.dk” Password = “1234” ConfirmPassword = “1234”	True	True	Positive test
CreateModel Email = “test@.dk” Password = “1234” ConfirmPassword = “1234”	True	True	Negative test, invalid email format
CreateModel Email = “test@test.dk” Password = “1234” ConfirmPassword = “12345”	True	True	Negative test, Password and ConfirmPassword does not match

ModelName: ParticipantModel

Input	Expected output	Actual output	Explanation
ParticipantModel: ParticipantEmail = “@stuff”	True	True	Negative test, incorrect email should fail.

ParticipantModel: ParticipantEmail = “stuff@stuff”	True	True	Negative test, incorrect email should fail.
ParticipantModel: ParticipantEmail = “@stuff.dk”	True	True	Negative test, incorrect email should fail.
ParticipantModel: ParticipantEmail = “stuff@stuff.dk”	True	True	Positive test, correct email should fail.

ModelName: QuestionModel

Method: ToTemplate

Input	Expected output	Actual output	Explanation

QuestionModel: Description: “Something about the following questions” Type : “Explanation” Required: False Options: []	explanation “Something about the following questions”	explanation “Something about the following questions”	Positive test
QuestionModel: Description: “Something about the following questions” Type : “Explanation” Required: True Options: []	explanation “Something about the following questions”	explanation “Something about the following questions”	Test that the Required attribute is ignored when using type “Explanation”
QuestionModel: Description: “Something about the following questions” Type : “Explanation” Required: False Options: [“a”, “b”]	explanation “Something about the following questions”	explanation “Something about the following questions”	Show that options are ignored when using type “Explanation”
QuestionModel: Description: “” Type : “Explanation” Required: False Options: []	explanation “”	explanation “”	Show that empty descriptions are valid
QuestionModel: Description: “Some text question” Type : “Text” Required: False Options: []	text “Some text question”	text “Some text question”	Positive test
QuestionModel: Description: “Some text question” Type : “Text” Required: True Options: []	mandatory text “Some text question”	mandatory text “Some text question”	Show that mandatory will appear in template if required property is true
QuestionModel:	text “Some text	text “Some text	Show that options are

Description: “Some text question” Type : “Text” Required: False Options: [“a” , “b”]	question”	question”	ignored when type is text
QuestionModel: Description: “Some checkbox question” Type : “Checkboxes” Required: False Options: [“a” , “b”]	checkboxes “Some checkbox question” [“a” , “b”]	checkboxes “Some checkbox question” [“a” , “b”]	Positive test Also tests Radiobuttons (as same code is used except type is “select” for radio buttons)
QuestionModel: Description: “Some checkbox question” Type : “Checkboxes” Required: False Options: []	checkboxes “Some checkbox question” []	checkboxes “Some checkbox question” []	Show that option list can be empty, even though this might be invalidated by the server
QuestionModel: Description: “Some question” Type : “Unknown” Required: False Options: []	<whitespace only>	Failure: description printed out anyways **FIXED**	Show that unknown question types are ignored in this model, and that only a string with whitespaces appear
QuestionMode: Description : NULL Type: “Explanation” Required : False Options: []	NullReference Exception	Failure: description was interpreted as an empty string **FIXED**	Show that there should be a failure when the description is null

Helpers

HelperName: EncryptionHelper
Method: Encrypt & Decrypt

Input	Expected output	Actual output	Explanation

String: “Test”	true	true	Take a string encrypt it, then decrypt and compare it with the original string.

HelperName: ParticipantHelper
Method: UpdateList

Input	Expected output	Actual output	Explanation
Invitee: Email = “stuff@stuff.dk” Name = “John Johnsson”	True	True	Positive test, Takes a list and updates it with another list.

Method: GetParticipant

Input	Expected output	Actual output	Explanation
participant = "stuff@stuff.dk"; ParticipantModelList: ParticipantEmail = "stuff@stuff.dk"	True	True	Adds a participant to the participant model, retrieves it and compares with the added participant.

HelperName: TemplateConversionHelper
Method: IsQuestion

Input	Expected output	Actual output	Explanation
q[0]	true	true	Positive test

q[10]	true	true	Positive test multiple digit id
q[]	false	false	No id given
q[wa]	false	false	Non-digit id

Method: IsOption

Input	Expected output	Actual output	Explanation
q[10]o[1]	true	true	Positive test
q[1]o[]	false	false	No option id given
q[]o[1]	false	false	No question id given

Method: IsType

Input	Expected output	Actual output	Explanation
qt[10]	true	true	Positive test

Method: IsRequired

Input	Expected output	Actual output	Explanation
req[10]	true	true	Positive test

Method: IsValidTitleContent

Input	Expected output	Actual output	Explanation
Something Something	true	true	Positive test
Something?	false	false	Negative test

Method: QuestionId

Input	Expected output	Actual output	Explanation
qt[10]	10	10	Positive test
q[01]	1	1	Positive test zero lead

Method: Convert

Input	Expected output	Actual output	Explanation
FormCollection: (format key->value): “title” -> “some stuff” “description” -> “Foo” “q[1]” -> “Hello World” “qt[1]” -> “Radio” “q[1]o[0]” -> “Hello” “q[1]o[1]” -> “World” “req[1]” -> “” “q[2]” -> “Bar” “qt[2]” -> “Text”	template some_stuff = begin explanation “Foo” mandatory select “Hello World” [“Hello” , “World”] text “Bar” end	template some_stuff = begin explanation “Foo” mandatory select “Hello World” [“Hello” , “World”] text “Bar” end	Positive test

Controllers

H.3 Integration test

Category	Input	Expected output	Actual output	Fixed
Authorization				
	Visiting /Survey/ while not logged in	Being returned to the login page	Being returned to the login page	
	Visiting /Home/ManageAcount/ while not logged in	Being returned to the login page	Being returned to the login page	
	Visiting /Home/Create while logged in	Being returned to the welcome page	**Seeing the Create User page**	✓
	Visiting /Home/Create while not logged in	Seeing the Create User page	Seeing the Create User page	
	Visiting /Survey/ while logged in	You see a list of your surveys and actions you can perform on them. Explanation that you have no surveys if that is the case.	You see a list of your surveys and actions you can perform on them. Explanation that you have no surveys if that is the case.	
	Visiting /Survey/Create while not logged in	Being returned to the login page	Being returned to the login page	
	Visiting /Participant/Index/xxx while not logged in	Being returned to the login page	Being returned to the login page	
	Visiting /Participant/Index/xxx on a survey you don't own	Error message that you don't have access to this survey	**Error message, "Sorry, an error occurred while processing your request."**	✓
	Visiting /Participant/SendMailEveryone/xxx while not logged in	Being returned to the login page	Being returned to the login page	
	Visiting /Participant/SendMailEveryone/xxx on a survey you don't own	Error message that you don't have access to this survey	**Error message, "Sorry, an error occurred while processing your request."**	✓
	Visiting /Participant/SendEmail/xxx?email=xxx@xxx.xx while not logged in	Being returned to the login page	Being returned to the login page	
	Visiting /Participant/SendEmail/xxx?email=xxx@xxx.xx on a survey you don't own	Error message that you don't have access to this survey	**Error message, "Sorry, an error occurred while processing your request."**	✓
	Visiting /Participant/Create/xxx while not logged in	Being returned to the login page	Being returned to the login page	
	Visiting /Participant/Create/xxx on a survey you don't own	Error message that you don't have access to this survey	**Being shown the submit form. It does not work when you submit, however.**	✓
Login				
	Attempt to log in using invalid user name (email) syntax	Failed logon, explanation that syntax is not valid	Failed logon, explanation that syntax is not valid	
	Attempt to log in with no password	Failed logon, explanation that password cannot be empty	Failed logon, explanation that password cannot be empty	
	Attempt to log in with non-existing user	Failed logon, explanation that the user doesn't exist	**Failed logon, explanation that the user/password combination doesn't exist**	✘
	Attempt to log in with existing user, wrong password	Failed logon, explanation that the user/password combination doesn't exist	Failed logon, explanation that the user/password combination doesn't exist	
	Attempt to log in with existing user, right password	Succesfull logon, welcome page being shown	Succesfull logon, welcome page being shown	
	Attempt to log in with "Remember password?" checked	Login cookie set to remain for 30 years	Login cookie set to remain for 30 years	
	Attempt to log in with "Remember password?" not checked	Login is session based	Login is session based	
Create user				
	Attempt to create a user with invalid email syntax	Creation failure, explanation that email is invalid	Creation failure, explanation that email is invalid	
	Attempt to create a user with a blank password	Creation failure, explanation that password cannot be blank	Creation failure, explanation that password cannot be blank	
	Attempt to create a user with password and confirm password not matching	Creation failure, explanation that passwords do not match	Creation failure, explanation that passwords do not match	
	Attempt to create a user that already exists	Creation failure, explanation that user already exists	**Creation failure, generic failure responce**	✓
	Attempt to create a new user that doesn't exist	Creation succes	**Creation succes, message that email was send (no such thing happens)**	✓
Manage account				
	Attempt to change password with wrong "old password"	No update, explanation that password was wrong	**No update, no explanation**	✓
	Attempt to change password with different new password and confirm new password	No update, explanation that password must match	**No update, no explanation**	✓
	Attempt to change with password with correct old password and matching new passwords	Succes	Succes	
	Delete account	Account is deleted	Account is deleted	

Category	Input	Expected output	Actual output	Fixed
Create survey				
	Click add new question	A new area for entering a new question appears	A new area for entering a new question appears	
	Change question type	Parameters change according to the question	Parameters change according to the question	
	Add an option	A new area for entering a new option appears	A new area for entering a new option appears	
	Create survey with invalid characters in survey name	Creation failure, explanation that the title contains invalid characters	Creation failure, explanation that the title contains invalid characters	
	Create survey with valid title and a combination of questiontypes and a variable amount of options per question	Succes	Succes	
Participants management				
	Clicking manage participants	Participants are shown	Participants are shown	
	Add participant with bad email syntax	Participant not added, explanation that syntax is wrong	Participant not added, explanation that syntax is wrong	
	Add participant with right email syntax	Participant is added	Participant is added	
	Send email to single participant	Email is sent	Email is sent	
	Send email to all participants	Email is sent to all participants	Email is sent to all participants	
	Remove participant	Participant is removed	Participant is removed	
Take survey				
	Taking a survey without all mandatory questions answered	Error that not all mandatory questions are answered	**Unfiltered exception with the first thing to fail + spelling error**	✓
	Taking a survey with all mandatory questions answered	Succes	Succes	
	Attempt to take a survey that has already been completed	Notice that it has already been completed and cannot be taken again	Notice that it has already been completed and cannot be taken again	
Miscellaneous				
	Clicking help link	Help page is shown	Help page is shown	
	Visiting /Home/Logout	User is logged out	User is logged out	
	Delete survey	Survey is deleted	Survey is deleted	
	Click statistics link	Statistics for the survey is shown	Statistics for the survey is shown	
	Viewing graphs for statistics that has options with no text	No text being used	**Graph breaks**	✓

Category	Input	Expected output	Actual output	Fixed
Create survey				
	Click add new question	A new area for entering a new question appears	A new area for entering a new question appears	
	Change question type	Parameters change according to the question	Parameters change according to the question	
	Add an option	A new area for entering a new option appears	A new area for entering a new option appears	
	Create survey with invalid characters in survey name	Creation failure, explanation that the title contains invalid characters	Creation failure, explanation that the title contains invalid characters	
	Create survey with valid title and a combination of questiontypes and a variable amount of options per question	Succes	Succes	
Participants management				
	Clicking manage participants	Participants are shown	Participants are shown	
	Add participant with bad email syntax	Participant not added, explanation that syntax is wrong	Participant not added, explanation that syntax is wrong	
	Add participant with right email syntax	Participant is added	Participant is added	
	Send email to single participant	Email is sent	Email is sent	
	Send email to all participants	Email is sent to all participants	Email is sent to all participants	
	Remove participant	Participant is removed	Participant is removed	
Take survey				
	Taking a survey without all mandatory questions answered	Error that not all mandatory questions are answered	**Unfiltered exception with the first thing to fail + spelling error**	✓
	Taking a survey with all mandatory questions answered	Succes	Succes	
	Attempt to take a survey that has already been completed	Notice that it has already been completed and cannot be taken again	Notice that it has already been completed and cannot be taken again	
Miscellaneous				
	Clicking help link	Help page is shown	Help page is shown	
	Visiting /Home/Logout	User is logged out	User is logged out	
	Delete survey	Survey is deleted	Survey is deleted	
	Click statistics link	Statistics for the survey is shown	Statistics for the survey is shown	
	Viewing graphs for statistics that has options with no text	No text being used	**Graph breaks**	✓